Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/6180--04-8800

Simulation Environment for Onboard Fire Network Model Version 1.0 — Theory Manual

THOMASZ A. HAUPT
DMITRY SHULGA
BHARGAVI SURA
SHRAVAN K. DURVASULA

Cooperative Computing Laboratory
Center for Advanced Vehicular Systems
ERC, Mississippi State University, Starkville, MS

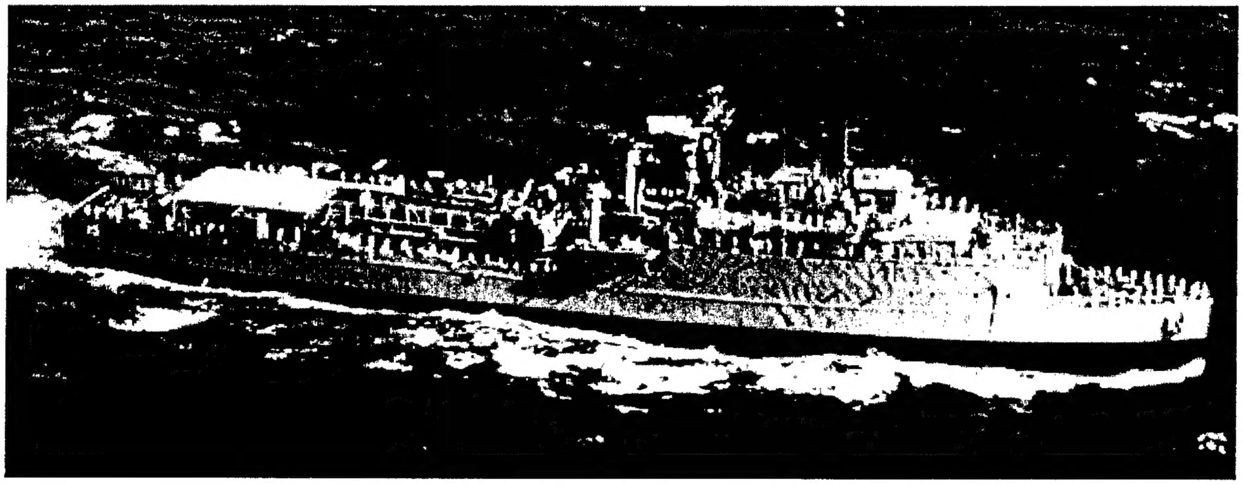
PATRICIA A. TATEM

ITT Industries, Advanced Engineering and Sciences Division
Alexandria, VA

FREDERICK W. WILLIAMS

Navy Technology Center for Safety and Survivability
Chemistry Division

May 12, 2004



REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 12 May 2004	**2. REPORT TYPE** Memorandum Report	**3. DATES COVERED** *(From - To)*
4. TITLE AND SUBTITLE Simulation Environment for Onboard Fire Network Model Version 1.0 — Theory Manual		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Thomasz A. Haupt,* Dmitry Shulga,* Bhargavi Sura,* Shravan K. Durvasula,* Patricia A. Tatem,† and Frederick W. Williams		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory, Code 6180 4555 Overlook Avenue, SW Washington, DC 20375-5320		**8. PERFORMING ORGANIZATION REPORT NUMBER** NRL/MR/6180--04-8800
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 800 North Quincy Street Arlington, VA 22217-5660		**10. SPONSOR / MONITOR'S ACRONYM(S)**
		11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*Cooperative Computing Laboratory, Center for Advanced Vehicular Systems, ERC, Mississippi State University, Starkville, MS
†ITT Industries, Advanced Engineering and Sciences Division, Alexandria, VA

14. ABSTRACT

The collaborative work of Hughes Associates, Inc. (HAI), the Naval Research Laboratory (NRL), and a group at Mississippi State University (MSU) resulted in developing a simulation system including Graphical User Interface (GUI) and visualization capabilities. The system is intended to provide real-time information to assist an emergency response team. The GUI must be relatively simple and straightforward to use. The design platform is Windows NT/XP. The general system design is developed with help of Unified Modeling Language (UML). The GUI is written with ANSI C^{++} programming language and Microsoft Foundation Classes (MFC) library. Windows API provides functionality for multithreading. OpenGL and mySQL are used for implementing visualizations and database, respectively. This report is organized as follows. The overall system design is presented in Section 2. Section 3 explains the structure of the code, and Section 4 discusses the database schema. Section 5 describes the process of populating the GUI data structures with data from the database, which is critical for the overall performance of the system. The information provided in Sections 2 through 5 is summarized in Section 6 that describes the resulting 3D model of the ship. Section 7 provides details of the runtime environment, including implementation of GUI elements for interactive control of the ship state and setting the network model parameters. The visualization of the network model output is described in Section 8. Finally, Section 9 summarizes the project.

15. SUBJECT TERMS

Modeling; Fire Modeling; Network Model; Simulation; Network

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Frederick W. Williams
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UL	36	**19b. TELEPHONE NUMBER** *(include area code)* 202-767-2476

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

CONTENTS

1.0 INTRODUCTION 1

2.0 SYSTEM DESIGN 2

- 2.1 Use Cases 2
- 2.2 System Components 3

3.0 CLASS HIERARCHIES 4

- 3.1 Class Identification 4
- 3.2 Geometry Classes 4
- 3.3 Ventilation and Fire Main Systems 5
- 3.4 Scene Classes 7
- 3.5 Main Scene 7

4.0 DATABASE STRUCTURE 8

- 4.1 Geometry 8
- 4.2 Ventilation System 9
- 4.3 Fire Main System 10
- 4.4 Simulation Data 10

5.0 DATABASE BUFFERING 11

- 5.1 Motivation 11
- 5.2 Classes and Structures 12
- 5.3 Scene Classes Dependence 14

6.0 3D MODEL OF THE SHIP 14

7.0 RUNTIME ENVIRONMENT FOR THE NETWORK MODEL 18

- 7.1 Multithreading 18
- 7.2 Input File Generator 18
- 7.3 Real-Time Simulations 19
  - 7.3.1 Status of Active Objects 19
  - 7.3.2 Simulation and Fire Parameters 20
  - 7.3.3 Running a Simulation 21
  - 7.3.4 Simulation Replay 22
  - 7.3.5 Comparison of Two Simulations 23

8.0 VISUALIZATIONS OF THE SIMULATION RESULTS 24

- 8.1 Color Mapping 25
- 8.2 Options Dialog Tab: Adjusting the Color Mapping 27
- 8.3 Smoke 28

9.0 SUMMARY 29

10.0 ACKNOWLEDGEMENTS 33

11.0 REFERENCES 33

SIMULATION ENVIRONMENT FOR ONBOARD FIRE NETWORK MODEL VERSION 1.0 – THEORY MANUAL

1.0 INTRODUCTION

The lifetime management of future naval vessels dictates its own requirements in addition to the phenomenological ones. During the ship's exploitation, it is necessary that all information accompanies the ship as related to its design, construction, operation, crew training and maintenance. These data need to be stored in a digital library or database and carried with the ship. The ability to perform fire modeling is needed throughout the ship life cycle.

Fire modeling is needed to evaluate ship designs and design philosophies to quickly arrive at an overall concept to meet required performance goals. As the design concept is refined, fire modeling is used to continue evaluation of ship vulnerability and to begin the process of ship operations.

During operation and crew training, the fire modeling requirements change drastically as the focus changes to damage control and recoverability efforts. In the design phases, the computational time of the various modeling techniques are not critical, but in the operational phases of a ship's lifetime, the model must provide information faster than fire-related events occur on board the ship while maximizing accuracy of prediction.

The collaborative work of Hughes Associates, Inc. (HAI), the Naval Research Laboratory (NRL), Havlovick Engineering Services and a group at Mississippi State University resulted in developing a simulation system including Graphical User Interface (GUI) and visualization capabilities. HAI provided a one-zone-based network model, which assumes that the modeled environment in each compartment can be represented by one set of physical variables as opposed to multiple set zone or CFD models. As such, a network model is capable of modeling an entire ship and its ventilation system. Since the number of variables being solved for is minimized – one per compartment – a network model also has the potential for the fastest computations.

The system is intended to provide real-time information to assist the emergency response team. The GUI must be relatively simple and straightforward to use – there will be no time for clicking buttons and making out complex schemes and diagrams when a fire is advancing to ship's control room.

The next GUI requirement is that crew members do not have to possess any knowledge of fire physics or fire protection engineering. In general, none of the future users of the systems will be experts in fire protection or physics of fire. Furthermore, the system must not overwhelm the operator; only minimum information that is highly relevant or recommended for fire suppression activity must be shown.

The fire simulations produce sufficient information for making adequate conclusions about environment and ship object states including the physics and chemistry of the fire: temperature, pressure, visibility (smoke) and species concentrations (toxins and oxygen).

Manuscript approved April 14, 2004.

The design platform is Windows NT/XP. The general system design is developed with help of Unified Modeling Language (UML). The GUI is written with ANSI C++ programming language and Microsoft Foundation Classes (MFC) library. Windows API provides functionality for multithreading. OpenGL and mySQL are used for implementing visualizations and database respectively.

The rest of this report is organized as follows. The overall system design is presented in Section 2. Section 3 explains the structure of the code, and Section 4 discusses the database schema. Section 5 describes the process of populating the GUI data structures with data from the database, which is critical for the overall performance of the system. The information provided in sections 2 through 5 is summarized in section 6 that describes the resulting 3D model of the ship. Section 7 provides details of the runtime environment, including implementation of GUI elements for interactive control of the ship state and setting the network model parameters. The visualization of the network model output is described in section 8. Finally, Section 9 summarizes the project.

2.0 SYSTEM DESIGN

2.1 Use Cases

The developed system is intended to serve as a design tool, a tactical tool or a training tool depending on the configuration (Fig. 1). This defines three major actors: a ship designer, a ship operator and a trainee.

- For a ship designer, the system must provide a solid feedback on geometry, ducts and other elements, as coming from CAD systems. The designer must be able to analyze and modify ship structure.
- For a ship operator, the system must provide real-time predictions on possible effects of an onboard fire. In addition, the operator must also be able to change ship structure to study the effects of wartime damages and run different "what-if" scenarios for optimized and more effective fire suppression.
- Finally, for training purposes, a trainee must be able to access simulations stored in the database and compare effectiveness of various fire suppression strategies.

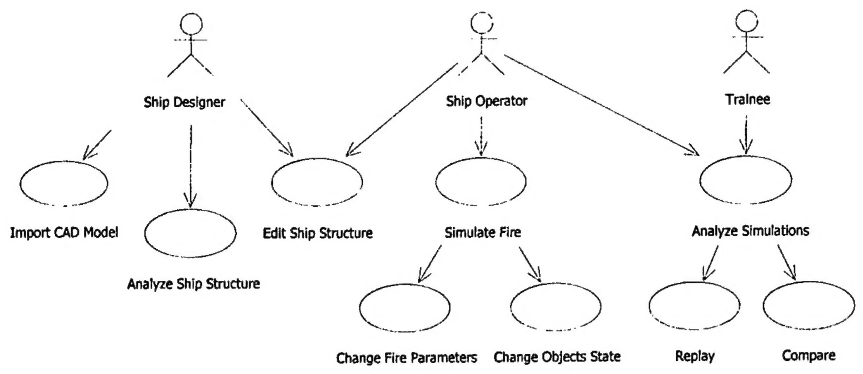


***Figure 1:** Use cases*

2.2 System Components

To satisfy the requirements, we have divided the overall system design into components as shown in Fig. 2.

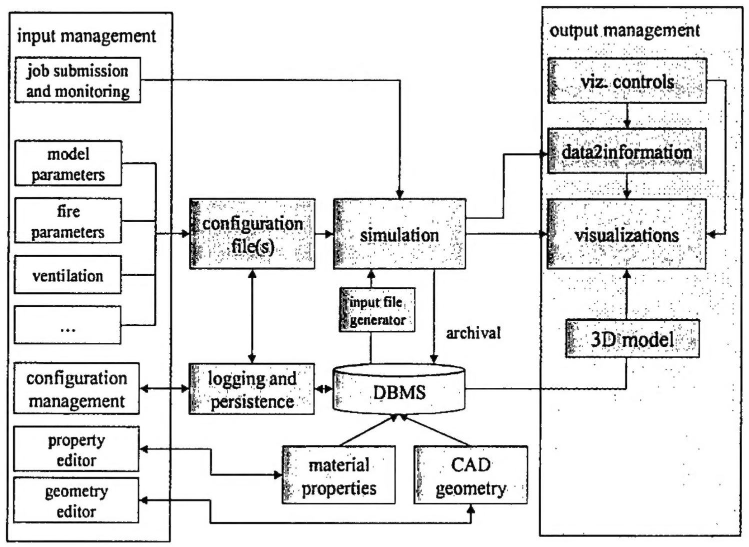


***Figure 2:** Architecture of the simulation environment for the onboard network fire model*

1. Input management
    - GUI for setting the fire parameters (location, size, type, etc.), and the state of the ship (state of the ventilation system, doors and hatches, initial concentration of species, etc.)
    - GUI for setting simulation parameters (such as duration of the simulation)
    - GUI for editing geometry and properties
2. Runtime environment for the network model
    - Database (DBMS) containing the geometry of the ship, material properties and results of previous runs
    - Input file generator that collects information from the GUI and the database and formats it according to the network model specification
    - Modules responsible for configuring, running and archiving results of the network model
3. Output management
    - 3D model of the ship
    - Real-time, 3D visualizations of the output: heat, smoke and toxins spread
    - GUI for replaying faster than real time results of previous runs
    - GUI for back-to-back comparisons of results coming from two different simulations

A more detailed discussion on the design process and literature review is given in [1].

3.0 CLASS HIERARCHIES

3.1 Class Identification

The object-oriented approach in design is an obvious choice today. It is more difficult to use than the function-based approach, but it produces a solid understanding of core processes that occur in the system and their initiators and participants. The system is represented by a complicated network of objects and their collaborations. A simple but powerful approach of noun extraction was applied to identify them.

A *ship* is built of *compartments*. A *compartment* is composed of *sides* each of which belongs to a *wall*. Each *side* is a set of *vertices*. A *wall* is built of two *sides*. It also includes *active elements* like *doors*, *hatches*, *scuttles*, a *ventilation system* with *fans* and *dampers* and a *fire main system* with *plugs* and *valves*. The *ventilation system* is built of *ventilation duct sections* that are composed of a pair of *ventilation nodes*. A *ventilation node* can be a *simple node*, a *fan* or a *damper*. The *fire main system* consists of *fire main sections* that are built of pairs of *fire main nodes*. In addition, a *fire main node* can be a *valve* or a *plug*. The activity of an object suggests its ability to be in multiple *states*. A *duct section* can have the following *states*: fake (virtual object not affecting simulation equations), disabled or enabled. A *door* can be fake, disabled, closed, opened or be a joiner. A *hatch*, *scuttle*, *fan*, *damper*, *valve* or *plug* can be fake, disabled, closed or opened.

The nouns are in italics and they define candidates for real classes. "Ship" is a general definition of the model so it should be ignored. Also "state" is not actually a real entity. It is a property of the object so it is also ignored as being represented as a class.

The denoted general idea about the system allows developing possible classes and hierarchies. All classes can be divided into two groups: scene classes and general classes. Scene classes can be split into two groups: geometry and systems. Geometry is defined by compartments, walls and sides. Systems include the ventilation system (doors, hatches, scuttles and ducts) and the fire main system.

In following sections, the design of classes of each group is considered separately in details.

3.2 Geometry Classes

A compartment represents a volume in space bounded by sides belonging to it. A wall is also defined by sides. Therefore, a side is a main visual element that will define geometric representation of the model.

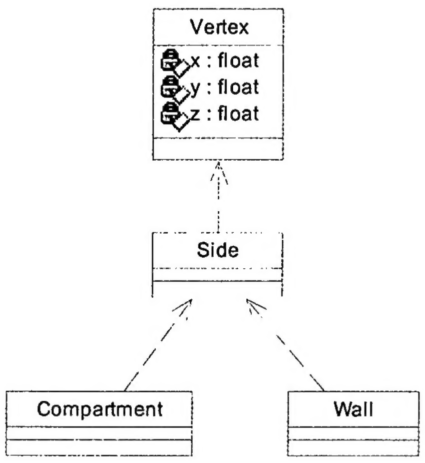


Figure 3: *geometry class associations*

A definition of a side is straightforward (see Fig. 3). In a sense of projection on one of three main planes, it is a flat polygon represented by a list of vertices fixed to four. Similarly, a compartment and a wall are lists of sides.

3.3 Ventilation and Fire Main Systems

The next observation is that a door, a hatch and a scuttle have the same semantic meaning – they are openings in a wall, floor or ceiling. As any opening does, they all have a position and size (Fig. 4).

Generally speaking, a ship may have not only above mentioned openings but also potentially any arbitrarily located object that can be represented as one and that appears anywhere due to a ship structure modification or outside impact-caused structural damage. The described structure satisfies such cases by adding a new class derived from the *Opening* class.

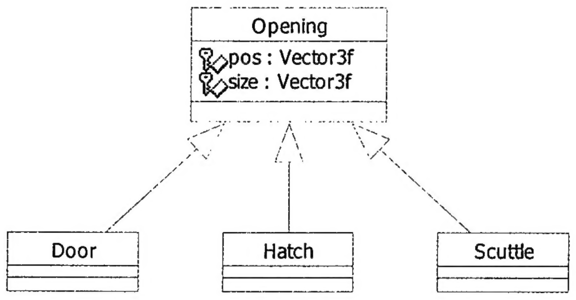


Figure 4: *Openings class hierarchy*

The next structural elements are ducts, e.g., a ventilation and a fire main duct. A duct is represented as a network of duct sections. Each section in turn is a pair of nodes or points in 3D space. Thus, a network is a collection of interconnected nodes. Connectivity of a given network is represented by duct sections.

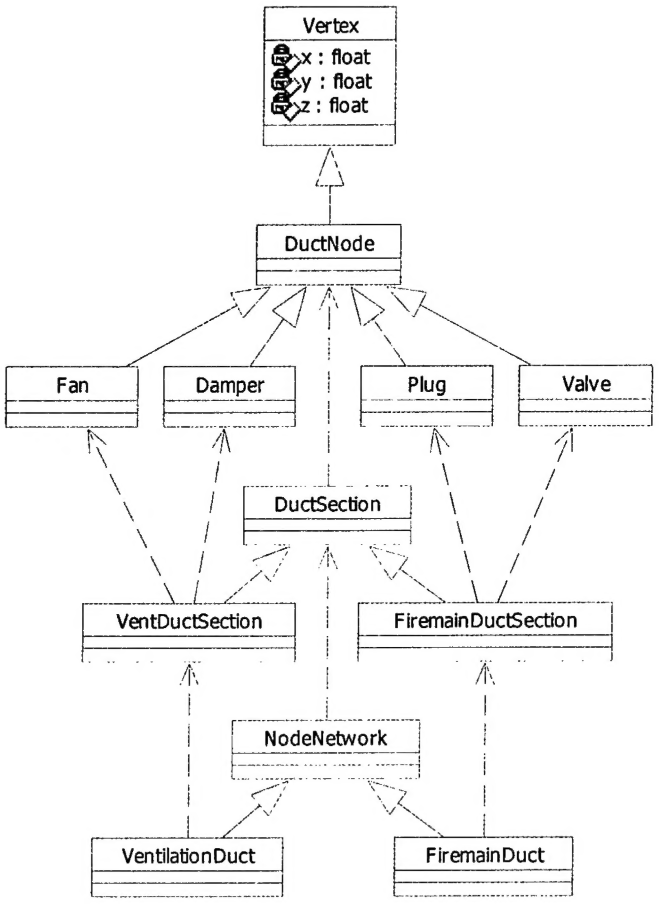


Figure 5: *Ducts class hierarchy*

A duct node may carry a meaning that is wider than just a point. It may possess some characteristics or behavioral attributes that may affect a network it belongs to. Nodes of a ventilation duct can be fans or dampers. Nodes of a fire main system can be plugs and valves (Fig. 5).

Further extension of this class family is also easy. It can be done by deriving from class bases, namely the *DuctNode*, the *DuctSection* and the *NodeNetwork*.

3.4 Scene Classes

The previously considered classes are not complete, hence not ready for explicit rendering. They should be converted into classes that can be shown on the screen, i.e. scene classes. A classical approach suggests using an abstract scene class that will represent a base for all other elements and encapsulate all necessary behavioral attributes like the ability to draw itself. This is the *SceneObjectBase.* The *SceneObjectBase* class allows the generalization of objects' representation and unification of the drawing process. Most often each object should have a position and a color as attributes as well as a drawing routing for calling by the render.

Not all of the ship elements are suited for rendering; i.e. drawing is meaningful only for ship geometry and systems. Also, some objects must be able to interact with a user and accept requests for changing their state. The object state may affect its appearance, physical or behavioral properties. Doors, hatches and scuttles are examples of such objects.

3.5 Main Scene

At this moment, it is possible to proceed with developing a manager of scene objects that is responsible for maintaining their creation, manipulation and finally release.

The previous section describes a range of scene classes. Among them, at least two groups should be distinguished: static and active objects. Thus, classes that do not have any dynamically updated properties will be children of the general scene base class, whereas for classes with active features (state, for one) a common parent will be created that will provide its descendants a functionality necessary for maintaining their dynamically updatable properties.

The system should also provide access to instances of the same class so that they can be treated differently from others. The *SceneObjectMngr* class, a manager of scene objects, serves these needs. It stores instances of the same class as a separate list, thus, always allowing identification and use of them independently. It extensively exploits C++ Standard Template Library that provides unprecedented flexibility, controllability and speed.

The *SceneBase* class is the hierarchy topmost class. It manages the whole rendering process and uses the *SceneObjectMngr* for scene objects management. The *SceneBase* provides a set of virtual functions for scene initialization, manipulation and drawing. The functions of that set can be overridden to enhance or change predefined behavior.

The complete class hierarchy is shown in Fig. 6.

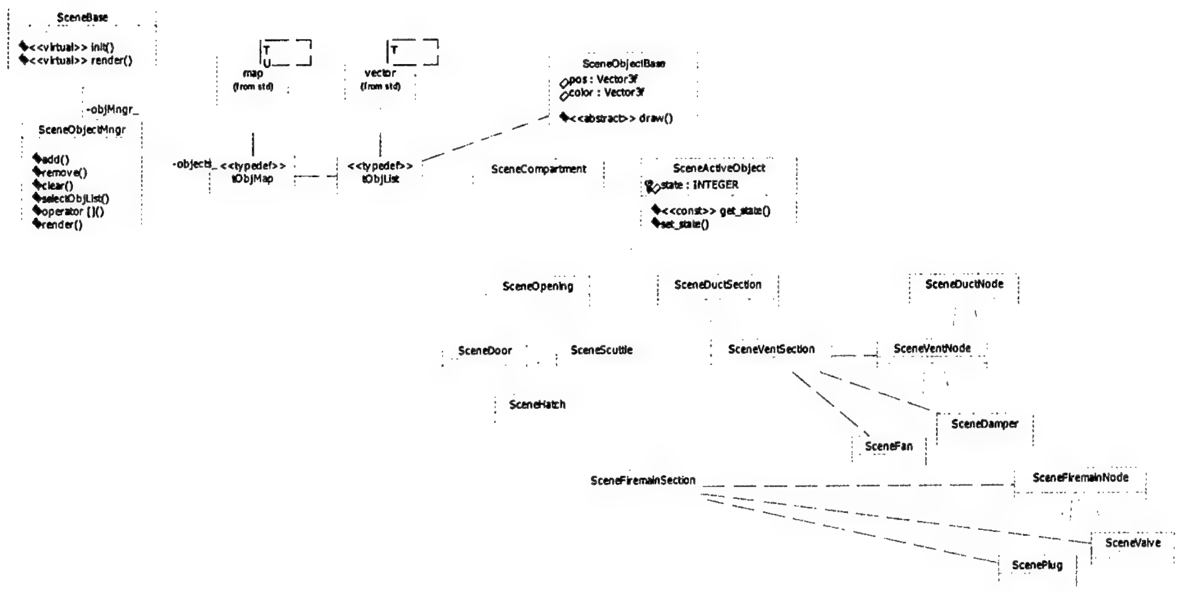


Figure 6: Complete scene class hierarchy

4.0 DATABASE STRUCTURE

4.1 Geometry

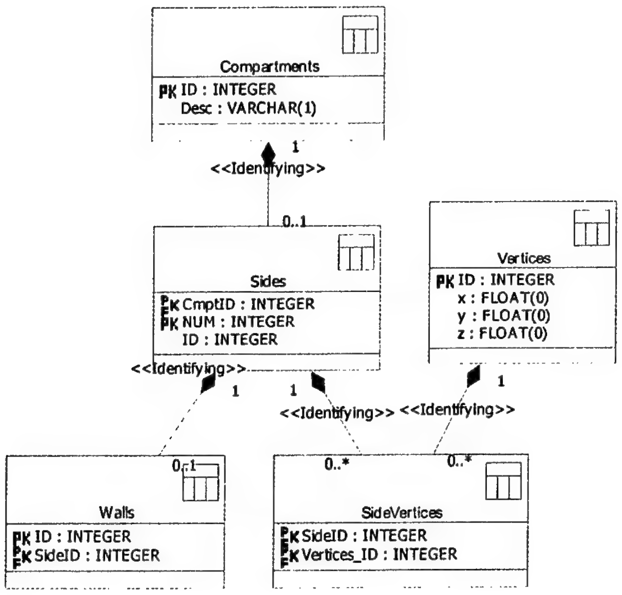


Figure 7: Geometry tables. Blue symbols PK and PKF in this and the following diagrams denote the table Primary Key and Foreign Primary Key respectively.

The *Compartment* represents a rather simple class that does not carry much information so far. Actually, the only additional field besides id is a description field. As mentioned before, each compartment is composed of sides which in turn are represented as a set of four vertices or points in 3D space. Each side also belongs to a wall. Generally, a wall consists of two sides and separates two compartments (Fig. 7).

4.2 Ventilation System

The ventilation system is represented by a ventilation duct and openings. A ventilation duct is a network of ventilation sections, each of which is a pair of ventilation nodes, i.e. points in 3D space. Theoretically, a section may consist of more than two nodes.

A node can be simple or complex. A complex node is a node that actively participates in the ventilation process. Currently, the only complex nodes are fans and dampers. The structure shown in Fig. 8 reflects the described relationship:

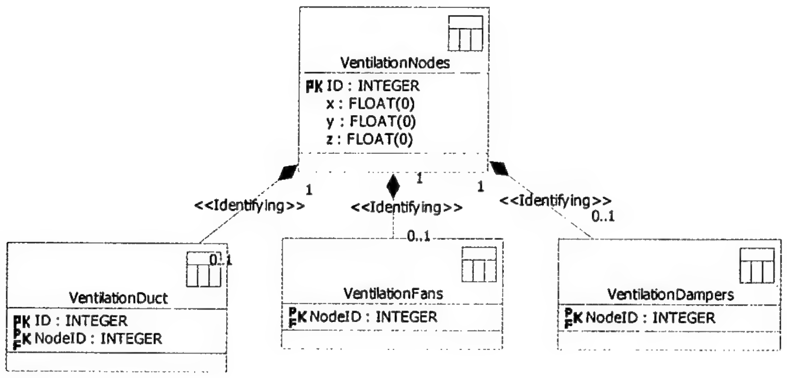


Figure 8: *Ventilation duct tables*

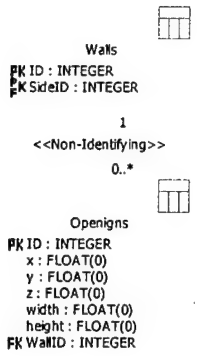


Figure 9: *Openings table*

Doors, hatches and scuttles have the same physical meaning so it is possible to store the information about them in one table. But for more convenience, three separate tables may be created. Each opening should possess knowledge of what wall it belongs to.

4.3 Fire Main System

The fire main system is very similar to the ventilation duct. It is also a network of node sections. The current active nodes of a fire main system are plugs and valves. The database table structure is shown in Fig. 10:

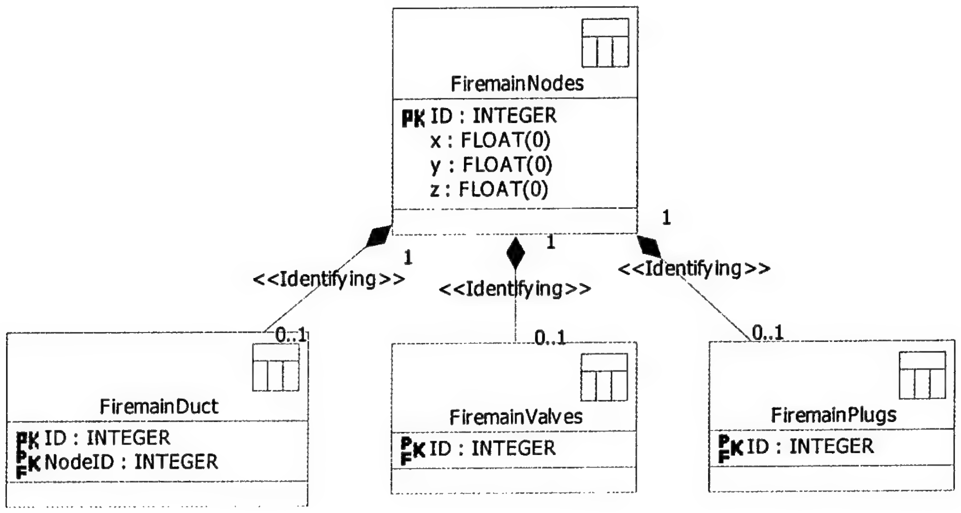


Figure 10: *Fire main tables*

4.4 Simulation Data

The predictions from the Network simulator need to be saved for future analysis and replays. The simulator produces a data block that contains the following scalar parameters (not exactly in the same order):

- Compartment temperatures
- Compartment pressures.
- Compartment O2.
- Compartment CO.
- Compartment soot.
- Compartment heat release.
- Duct node temperatures (not used).
- Duct node pressures (not used).
- Front surface temperatures (not used).
- Back surface temperatures (not used).
- Fire size (not used).

Compartment related data is combined into one table. Similarly, duct node data are placed into another table.

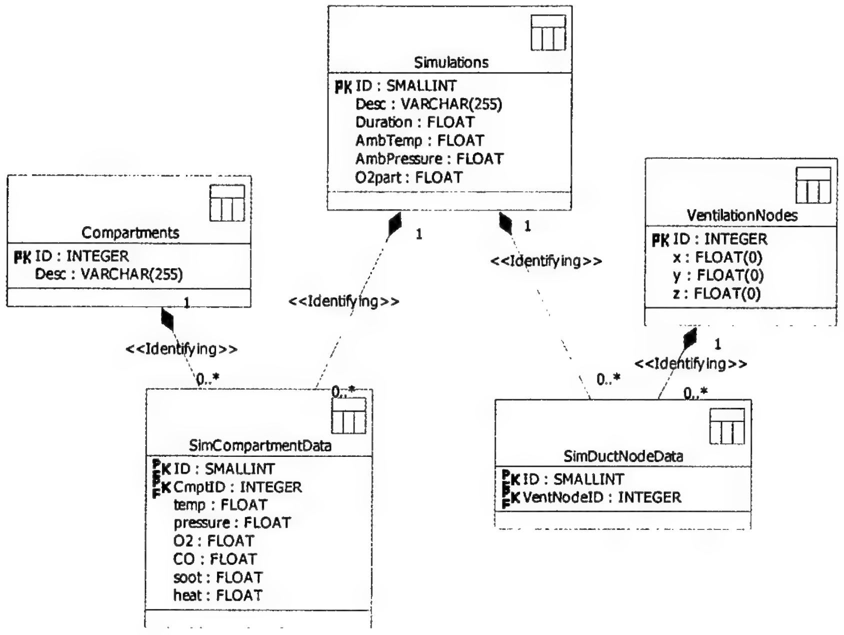


Figure 11: Tables for storing simulation data

Using these tables, it is very easy to access and study the simulation results in the scope of compartments (all or single), ducts, time or space.

5.0 DATABASE BUFFERING

5.1 Motivation

Scene classes described above are able to render themselves, but they still need to know where to render themselves on the screen, i.e. they need to know their coordinates. This information comes solely from the database. Database access is fast but still incomparably slower than access to data stored in the main computer memory. Since rendering is an extremely demanding process, the best performance highly depends on the amount of information to render and access information speed. Critical information should read, or be pre-buffered, from the database into the main memory.

The most significant information is ship geometry. Due to complexity, a very quick access to all ducts, e.g. a ventilation duct, must also be provided. The next section presents the data structures for storing and manipulating mentioned types of the data.

5.2 Classes and Structures

As referred before, some elements of the geometry are composed of smaller units, e.g. vertices comprise a side and sides comprise a wall or a compartment. This general approach helps to create a class, the customization of which easily allows us to reflect described relationships.

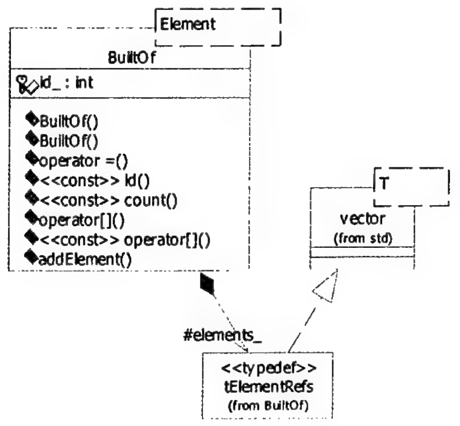


Figure 12: Generic class for representing a complex entity (for example, side is composed of vertices)

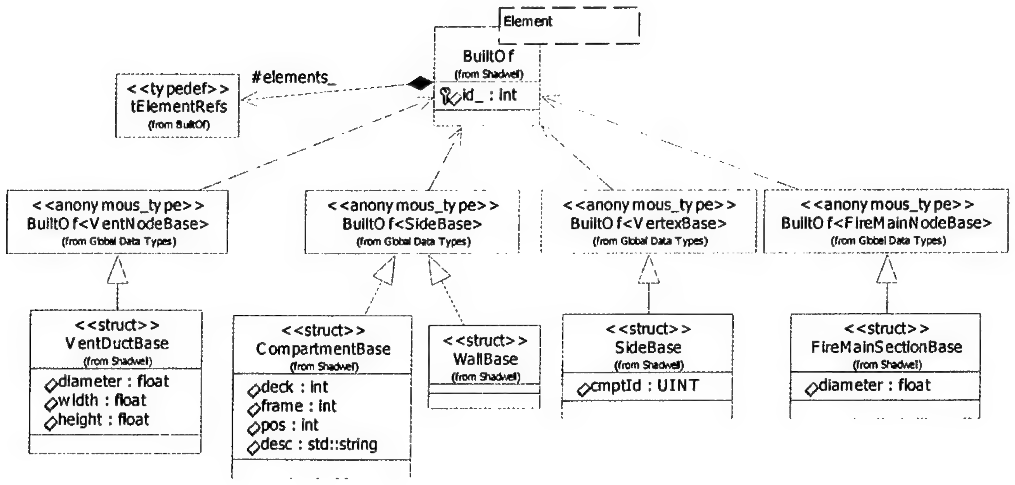


Figure 13: Geometry data storage class

The class represents a wrapper around an array of pointers to instances of arbitrary classes. The class provides functionality to access interesting elements of the array. Using this generic representation, a data structure or class can be built that it will accommodate the information from the database (Fig. 12). This class serves as a base class for all complex geometry elements (Fig. 13).

A centralized class management increases code accuracy, efficiency and maintainability. The *BodyStructure* class serves as a depository of all geometric data (Fig. 14).

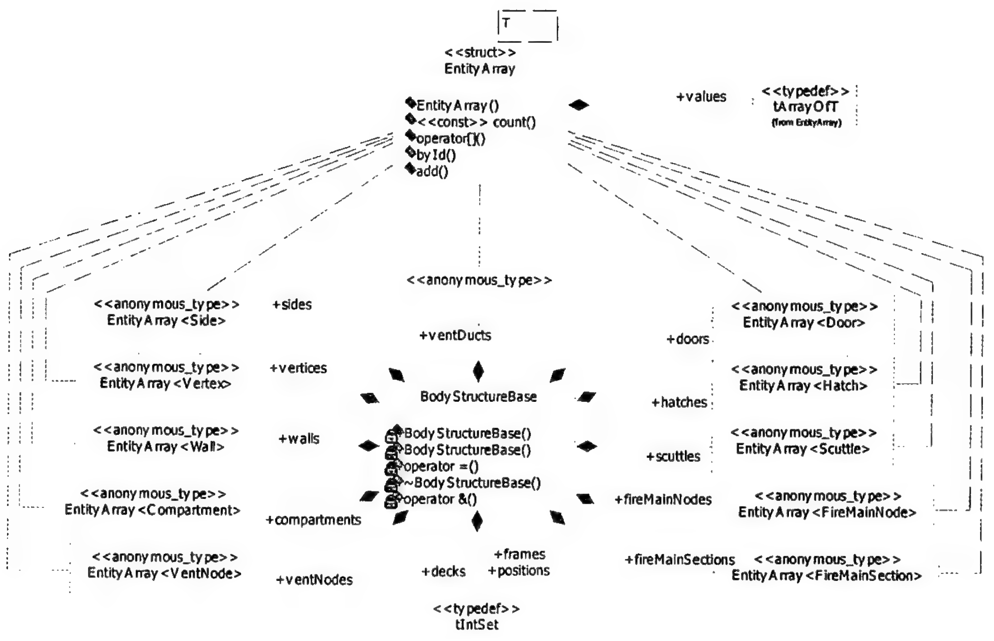


Figure 14: The BodyStructure class – a container for all ship data

The most important detail about the design is that it does not duplicate any data. The simplest unit of the geometry is a vertex. Vertices are read from the database as they are. A set of vertices defines a side, so a side has knowledge of them by creating an array of references to already created and loaded vertices. In turn, a wall or compartment contains references to sides. This is a very flexible and memory-efficient scheme that also is extensible and easily evolvable.

The described hierarchy also needs a very sophisticated loader. The loader can be designed to provide a generic functionality capable of reading different parts of the geometry data with just a few customization details since it is built with templates. A source of data is transparent for a loader; i.e. it uses a bridged connection, or interface, to access information (data bridging is described in later sections). After data reading, a loader creates necessary data interconnections by means of references.

5.3 Scene Classes Dependence

A motivation for data buffering was that scene objects must possess information on how to draw themselves. The classes described in the previous section are intended to provide such information. The dependence between them is straight forward – a scene class is associated with a corresponding data buffering class (Fig. 15).

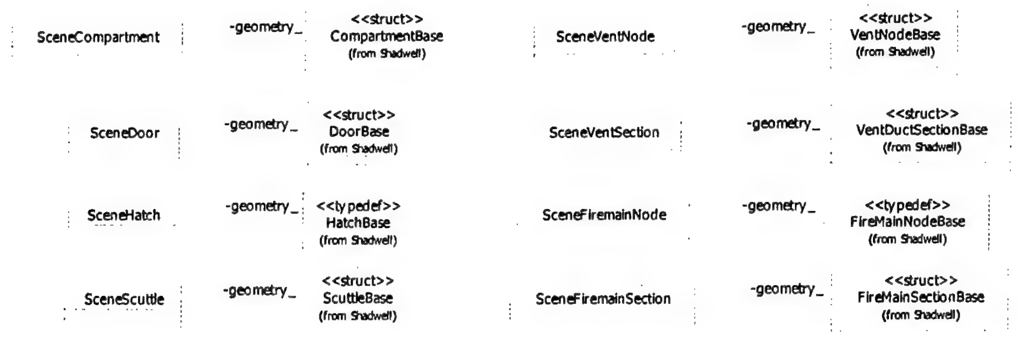

SceneCompartment
-geometry_
<<struct>> CompartmentBase (from Shadwell)
SceneVentNode
-geometry_
<<struct>> VentNodeBase (from Shadwell)
SceneDoor
-geometry_
<<struct>> DoorBase (from Shadwell)
SceneVentSection
-geometry_
<<struct>> VentDuctSectionBase (from Shadwell)
SceneHatch
-geometry_
<<typedef>> HatchBase (from Shadwell)
SceneFiremainNode
-geometry_
<<typedef>> FireMainNodeBase (from Shadwell)
SceneScuttle
-geometry_
<<struct>> ScuttleBase (from Shadwell)
SceneFiremainSection
-geometry_
<<struct>> FireMainSectionBase (from Shadwell)

Figure 15: *Relationships between scene and data buffering classes*

6.0 3D MODEL OF THE SHIP

The data collected in the buffering classes are used to build a 3D model of the ship ex-USS Shadwell/test area 688. The simplest visualization is a wire frame model shown in Fig. 16.

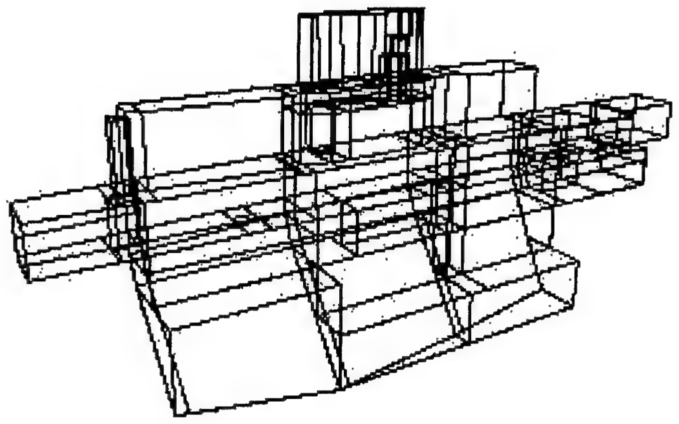


Figure 16: *Wire-frame model of the ship*

It can be zoomed and rotated to reveal the ship features of choice (Fig. 17).

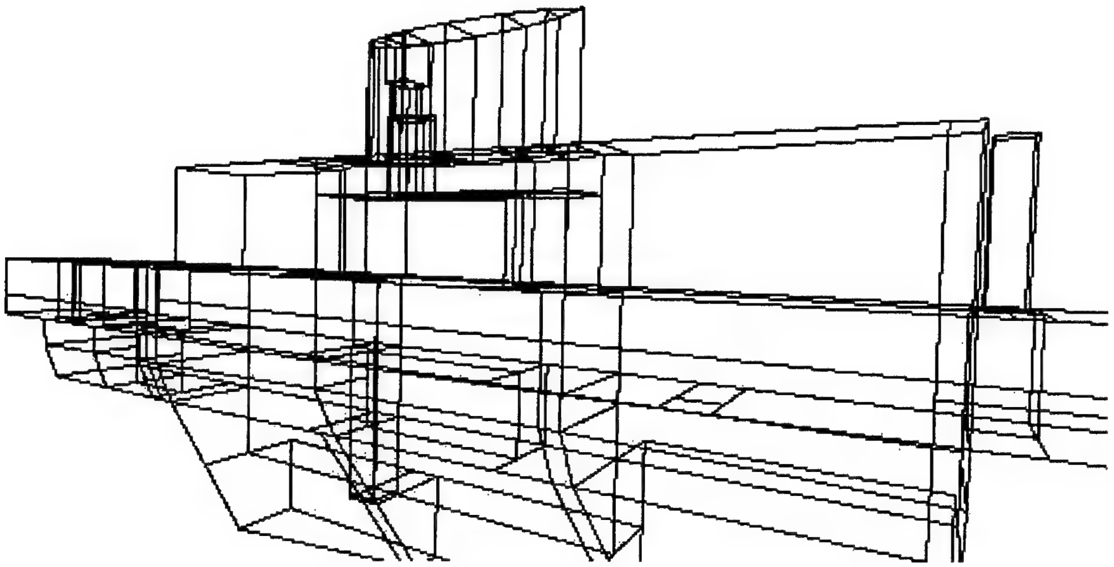


Figure 17: *Zoomed and rotated view of the wire-frame representation of the ship*

By showing bulkheads we provide a more realistic view of the ship (Fig. 18).

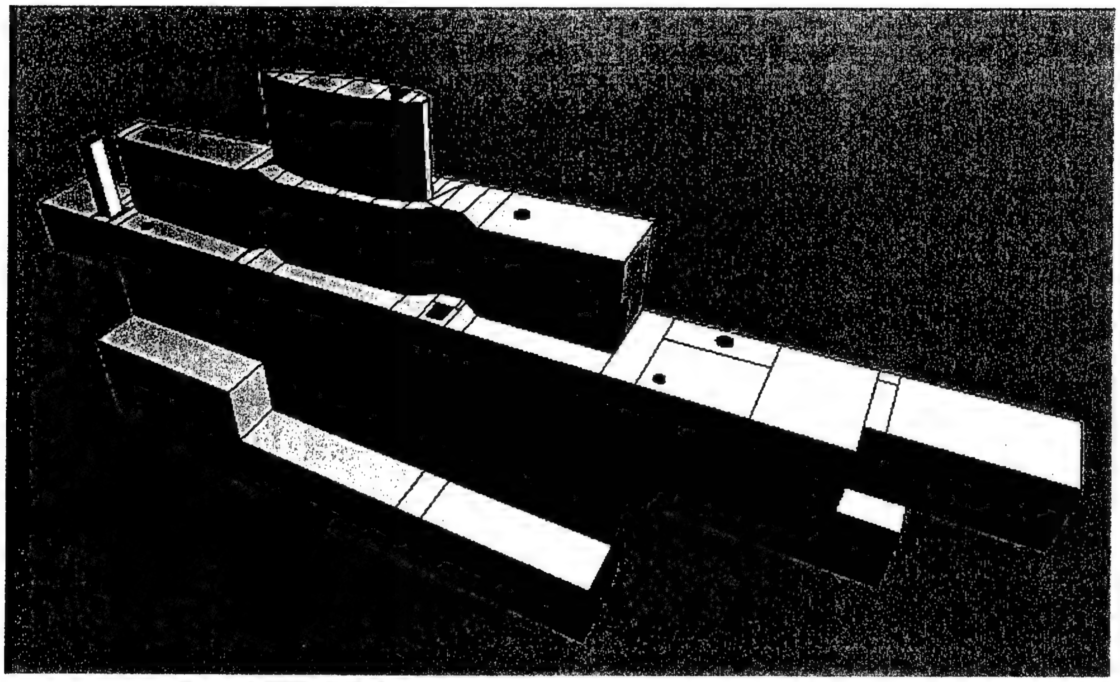


Figure 18: *3D solid model of the ship (doors, hatches and scuttles shown as well)*

However, we found it most useful to remove the front bulkheads that obstruct view into the inside of the ship. When the ship is rotated 180 degrees, the front bulkheads become back ones and vice versa. Therefore, the removal of the bulkheads is done on-the-fly depending on the position of the ship (Fig. 19).

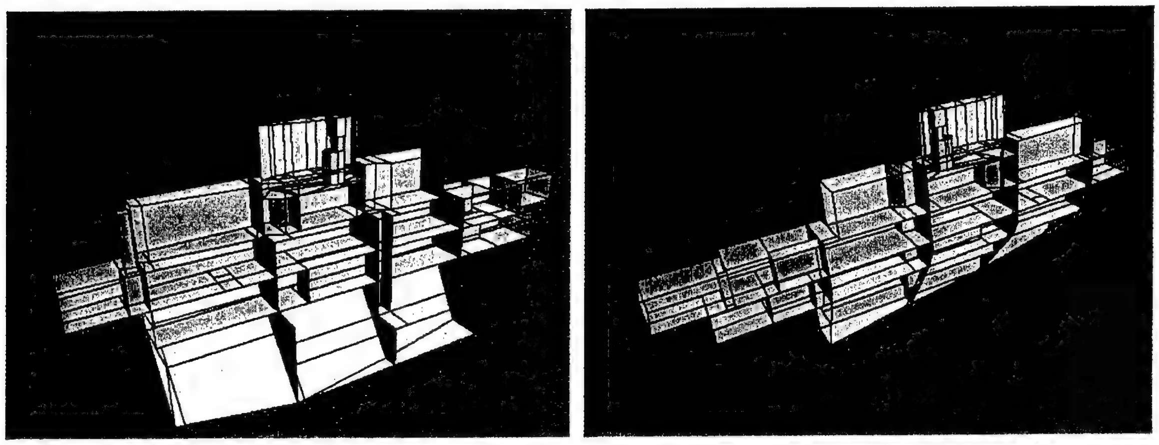


Figure 19: *3D model of the ship with the front bulkheads removed*

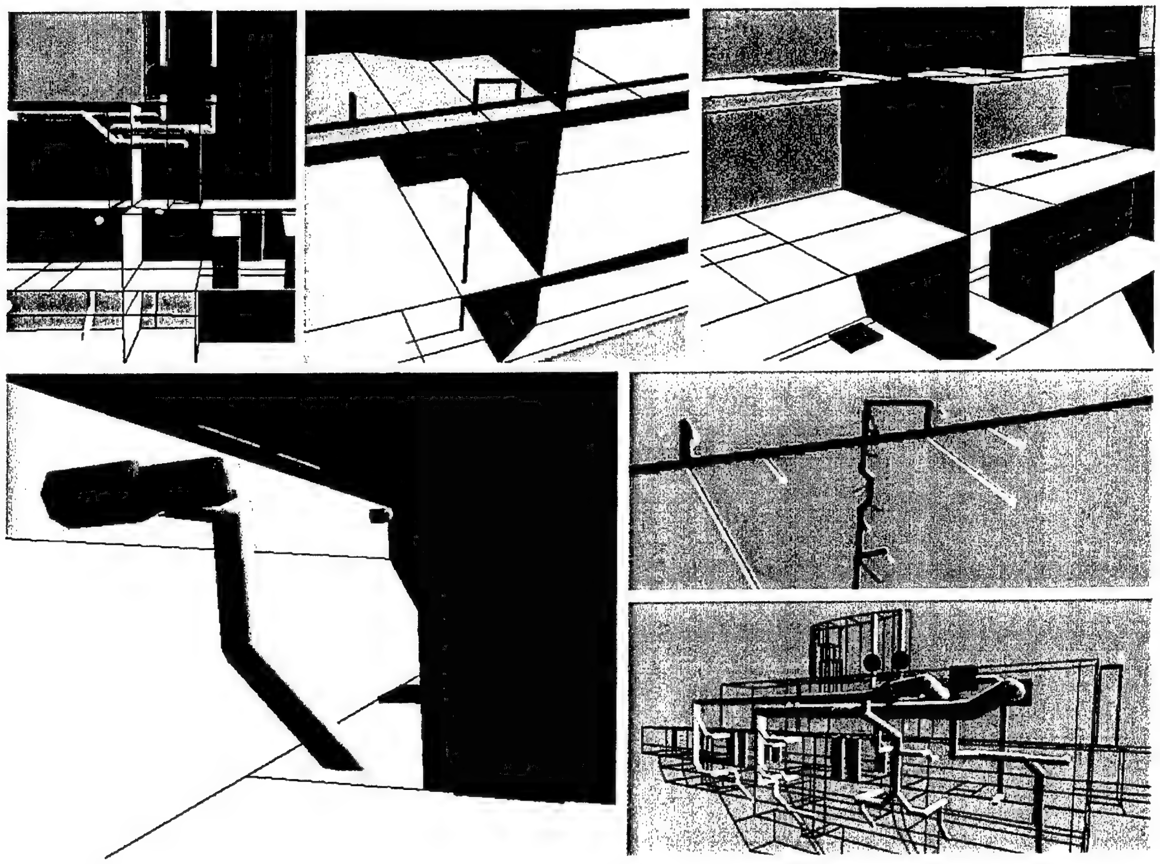


Figure 20: *Views of different ship elements*

With the basic geometry in place, we can add openings (doors, hatches and scuttles) as well as ship subsystems (ventilation and firemain). Those can be shown separately or in combination with other elements (Fig. 20). Finally, we provide a capability to inspect particular fragments of the ship, such as selected compartments or decks as shown in Fig. 21.

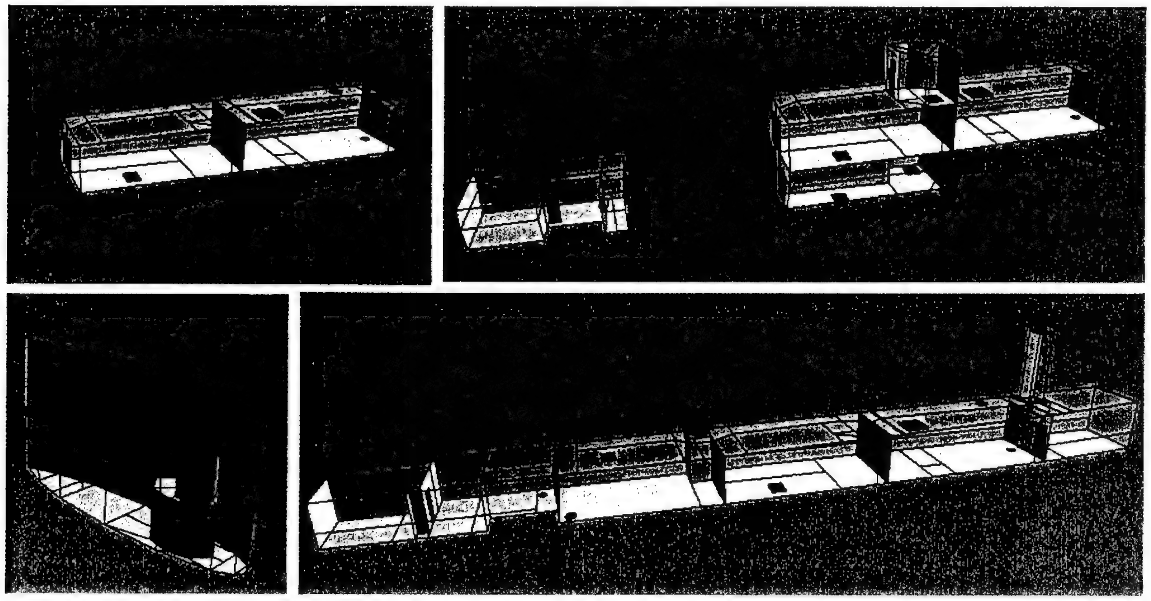


Figure 21: *Selected compartments*

A complete 3D model of the ship is shown in Fig. 22.

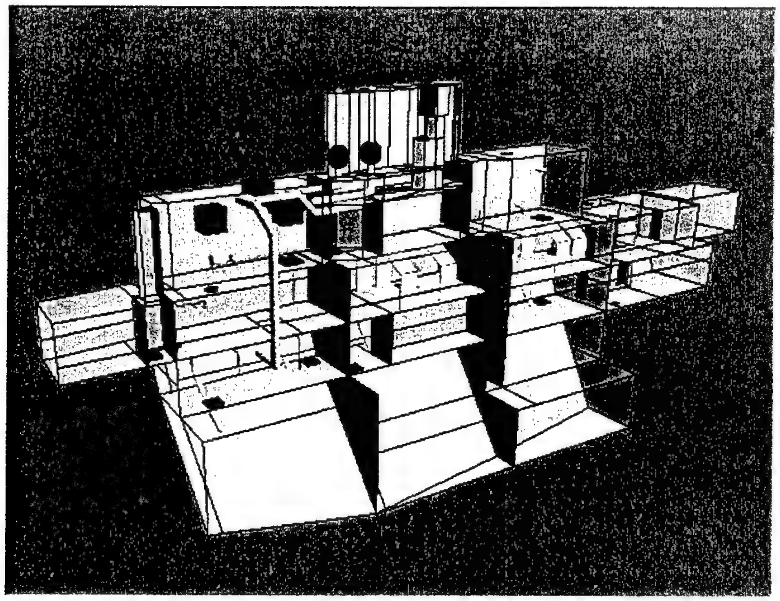


Figure 22: *The complete 3D model of the ex-USS Shadwell/test area 688*

7.0 RUNTIME ENVIRONMENT FOR THE NETWORK MODEL

7.1 Multithreading

The network model [2] developed by Hughes Associates and delivered to Mississippi State University as a MS Windows executable is run as a separate thread. There are several reasons for creating a multithreading environment. The network model and the rest of the system, in particular, real-time visualizations and controls, are competing for the CPU. Thus, the most important role of the runtime environment is to guarantee the user run-time control over simulation while allocating as much CPU for the simulation as possible. The amount of CPU allocated for the simulation critically influences its performance. On the other hand, the rendering functionality must be available for redrawing a ship model after each time step. Also, it is necessary to support the interaction with the user, which includes pausing, resuming and stopping simulation execution.

Multithreading under the Windows operating system can be achieved in different ways. Our implementation takes advantage of Windows native functions:

- *CreateThread* for a simulation thread properties initialization and its start.
- *CloseHandle* for releasing system resources allocated for a thread.
- *SuspendThread* for pausing or suspending a running simulation thread.
- *ResumeThread* for resuming execution of a suspended simulation thread.
- *TerminateThread* for exiting from or forced termination of a running simulation thread.

7.2 Input file generator

The network model is a standalone application written in FORTRAN 95 that accepts input in the form of a namelist file and produces formatted text output [3]. A namelist file, which is a standard FORTRAN language feature, comprises lines of formatted text data. Simplifying the FORTRAN standard, a definition of the format of the namelist file is as follows:

```
NAMELIST /namelist-group-name/ [attribute=value[, attribute=value…]]
```

Each namelist-group-name defines its own set of attributes. For example, junctions – objects connecting two others (openings and duct sections) – in the Network model are defined as:

```
&JUNC id=5, kloss=2.04, height=-2.66,-2.66, span=1.88,1.88,
area=0.95, location= 3,4, orientation=4,
bidirectional=.TRUE./Door 1' Control-NAV
```

The code accepts namelist statements with the following tags:

- EXEC – general simulation parameters.
- FIRE – fire source parameters.
- JUNC – junction parameters.
- CTRL – control element parameters.
- COMP – compartment data.
- SURF – surface data.
- MTRL – compartment walls material data.
- CMPN – component of a material.
- RDCT – ventilation system.
- RNOD – ventilation system nodes.
- RFAN – ventilation fan parameters.
- CURV – an item of tabular data.

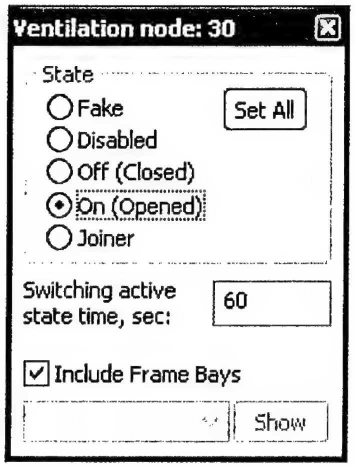
It is the simulation environment's responsibility to generate the input file for the network model, that is, to generate the namelist statements. The data come from the database describing the ship geometry and GUIs through which the user specifies the state of the ship and simulation parameters.

7.3 Real-Time Simulations

Creation of a new simulation begins with the definition of states of the ship's active objects (e.g., doors and fans), setting simulation parameters (e.g., duration of the simulation and ambient environment parameters) and fire propagation parameters (e.g., number of fire sources and their strength) though the GUI.

7.3.1 Status of Active Objects

Each active object may be in several different states depending on its type. Any object can be declared as *fake* (not related to any physical object included in the simulation area) or *disabled* (its state cannot be changed; for example, one cannot open an external hatch while the submarine is submerged). An opening can be either in an *off* (or *closed*) state or an *on* (or *opened*) state. A door also can be in a *joiner* state (partially closed). Using the state edit dialog, shown in Fig. 23, the user can set the state of each active


Ventilation node: 30

State: Fake Disabled Off (Closed) [✓ On (Opened)] Joiner

Switching active state time, sec: 60

[✓] Include Frame Bays

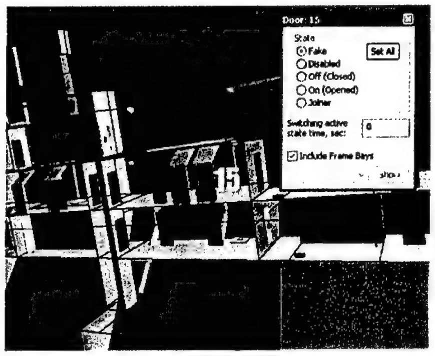
Figure 23: *The Object State Edit dialog*

object separately, or set all active objects to the same state by selecting a sought state and clicking on the *Set All* button.

The network model does not accept the status changes of an active object at runtime (this feature will be implemented at a later time). However, a delayed change of status can be declared prior to the model execution. For example, in Fig. 23, the ventilation node 30 (fan) is scheduled to change its status from on to off after 60 seconds of the simulation time. The change of state always means a transition from one state to the opposite: e.g., from open to close or from off to on.

Fig 24: Frame Bays

The final feature available from the Object State Edit Dialog is the ability to include or exclude the Frame Bays in the simulation. Frame Bays are a test area 688 specific ventilation system feature. Frame Bays were added to the 688 test area of the ex-USS Shadwell to simulate openings between the decks and hull of a submarine through which heat, smoke, and other combustion products can travel. Frame Bays are displayed as vertical flat ventilation sections. For example, frame bays connecting the combat system room (1-75-2) and the torpedo room (3-74-2) are shown in Fig. 24.



(● Fake) Disabled; Off (Closed); On (Opened); Joiner
[✓] Include Frame Bays

Figure 25: fake doors

An example of a "fake" object is a side door on deck 2. It physically exists in the 688 test area; is present in the CAD drawings, and is therefore transferred to the database. This door is always closed during tests in the 688 area because the 688 test area is intended to simulate a submarine and no submarine has side doors. By introducing "fake" objects we preserve the accuracy of the ship representation (with respect to CAD drawings), while hiding them from the simulation operator (Fig. 25).

A ship model can have hundreds of active elements. It is impractical to force a user to set all of them for each new simulation. To overcome this problem, several default modes are provided; each of which will define a unique set of states for all active elements. For the currently used ship model, there are three predefined modes: *Recirculation*, *Snorkel* and *Pier-side*. By default, the system is set to the most frequently used one.

The status of the active object is passed to the model as elements of the namelist file.

7.3.2 Simulation and Fire Parameters

Fig. 26 shows the GUI that allows the user to set simulation and fire parameters as well as to control the simulation (start, pause-resume or stop).

Simulation parameters include simulation description and name of the name list file created for the particular run, the requested duration of the simulation, external conditions (ambient temperature and pressure), and initial concentration of oxygen (assumed to be constant throughout the entire ship).

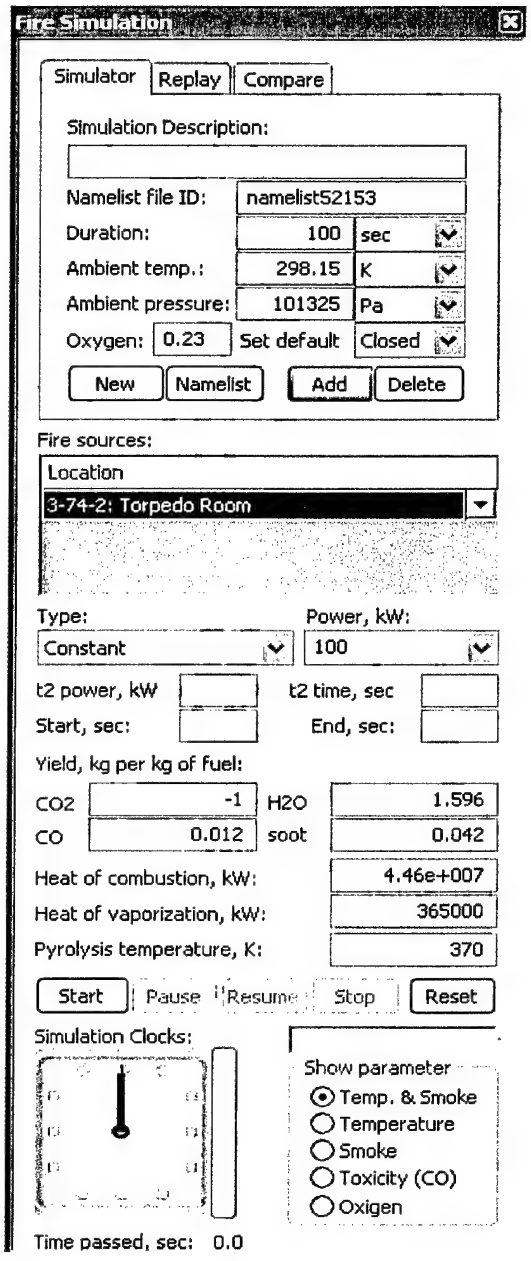
The description and name of the namelist file is used for identification of the simulation, used for subsequent replay or comparison.

The "New" button resets the current settings to their default values. The "Namelist" button shows the actual contents of the generated namelist file. This feature is targeted only for expert users and for debugging purposes.

Each of these data fields (upper part of the GUI shown in Fig. 26) has a default value, including a description, which will be set into a name of the input namelist file in case no description is given.

The controls in the central part of the GUI allow definition of fire source(s). Each fire source has a unique set of parameters that includes location (to be selected from a drop list), fire type (constant, t^2 fire and tabular), power, starting and ending times, fuel parameters and others, as shown in Fig. 26. By default, a fire source is constant in time with a power of 100 kWatts.

A simulation can have several fire sources each of which may have different settings. A new fire source is created by clicking the "Add" button and can be removed by using "Delete" button.



Simulation Description:

Namelist file ID: namelist52153

Duration: 100 sec

Ambient temp.: 298.15 K

Ambient pressure: 101325 Pa

Oxygen: 0.23 Set default Closed

Fire sources:

Location: 3-74-2: Torpedo Room

Type: Constant

Power, kW: 100

t2 power, kW

t2 time, sec

Start, sec:

End, sec:

Yield, kg per kg of fuel:

CO2	-1	H2O	1.596
CO	0.012	soot	0.042

Heat of combustion, kW: 4.46e+007

Heat of vaporization, kW: 365000

Pyrolysis temperature, K: 370

Show parameter: [✓ Temp. & Smoke] Temperature Smoke Toxicity (CO) Oxigen

Time passed, sec: 0.0

Figure 26: *The Fire Simulation dialog window*

7.3.3 Running a Simulation

All this information (together with the ship geometry stored in the database, and state of the active objects) is used to generate an input file for the network model (i.e., the namelist file). The creation of the namelist file occurs when the user clicks on the "Start" button (Fig. 27). If it succeeds, the system starts the

Network model in the separate thread and begins processing its output step by step. The user may pause or suspend this process, resume execution of a suspended simulation or stop it by clicking on the corresponding button (Fig. 28).

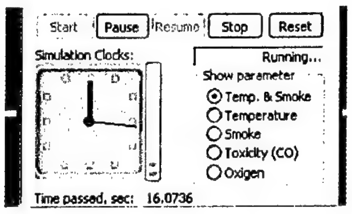


Figure 27: *After clicking the "Start" button the namelist file is created, and the status line is set to "Preparing ..." If the incorrect data are entered (no fire source defined, duration is not an integer number, etc) the status line is set to "Error..."*

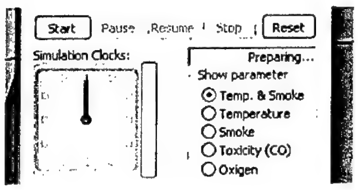


Figure 28: *After a successful generation of the input file the simulation begins. The simulation time is shown using an analog clock, a digital clock as well as a progress bar. The digital clock provides exact time. A digital clock make it easy to compare the rate of simulations with the real time while the progress bar gives an estimate how much time is left until the end of simulation.*

7.3.4 Simulation Replay

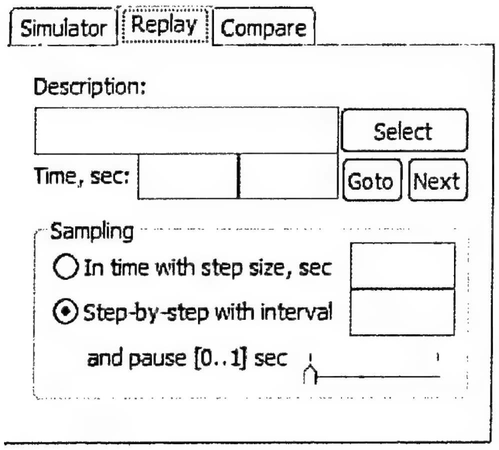
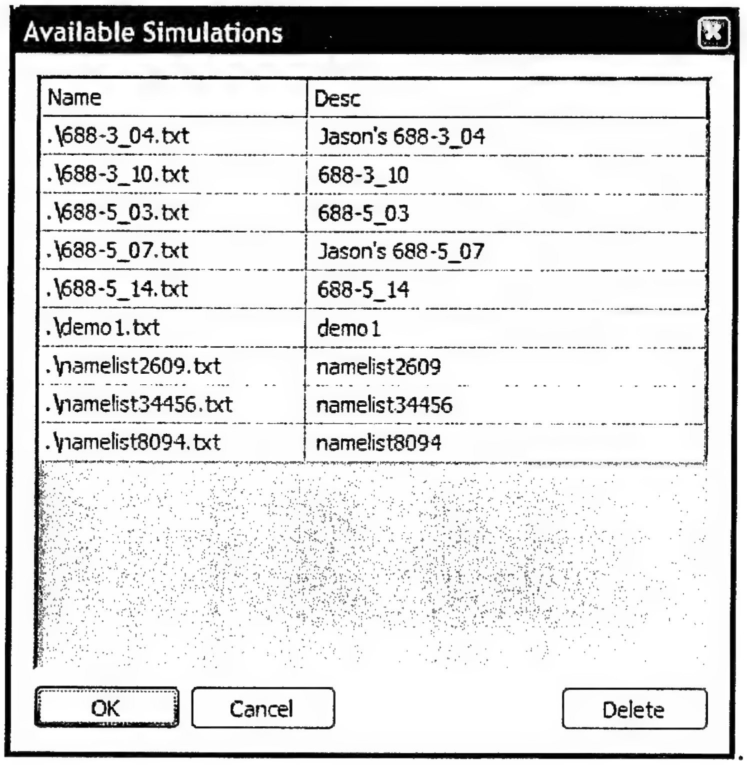


Figure 29: *The Replay tab of the Fire Simulation dialog window*

Since all simulations (both settings and results) are stored in a database, each simulation can be replayed at later time (Fig. 29). Replaying means that the results are read from an existing file and not generated in real time.

To replay a simulation, the user must select it from the list of available simulations (Fig. 30) which contains the names of namelist files and the descriptions of the simulations. After a simulation is chosen, the selected namelist file is parsed and the system sets the ship's objects into the appropriate states. To start the replay, the user must press the *Start* button. The functionality of the rest of the buttons is the same as described previously.



Name	Desc
.\688-3_04.txt	Jason's 688-3_04
.\688-3_10.txt	688-3_10
.\688-5_03.txt	688-5_03
.\688-5_07.txt	Jason's 688-5_07
.\688-5_14.txt	688-5_14
.\demo1.txt	demo1
.\namelist2609.txt	namelist2609
.\namelist34456.txt	namelist34456
.\namelist8094.txt	namelist8094

***Figure 30**: the Available Simulations dialog Window*

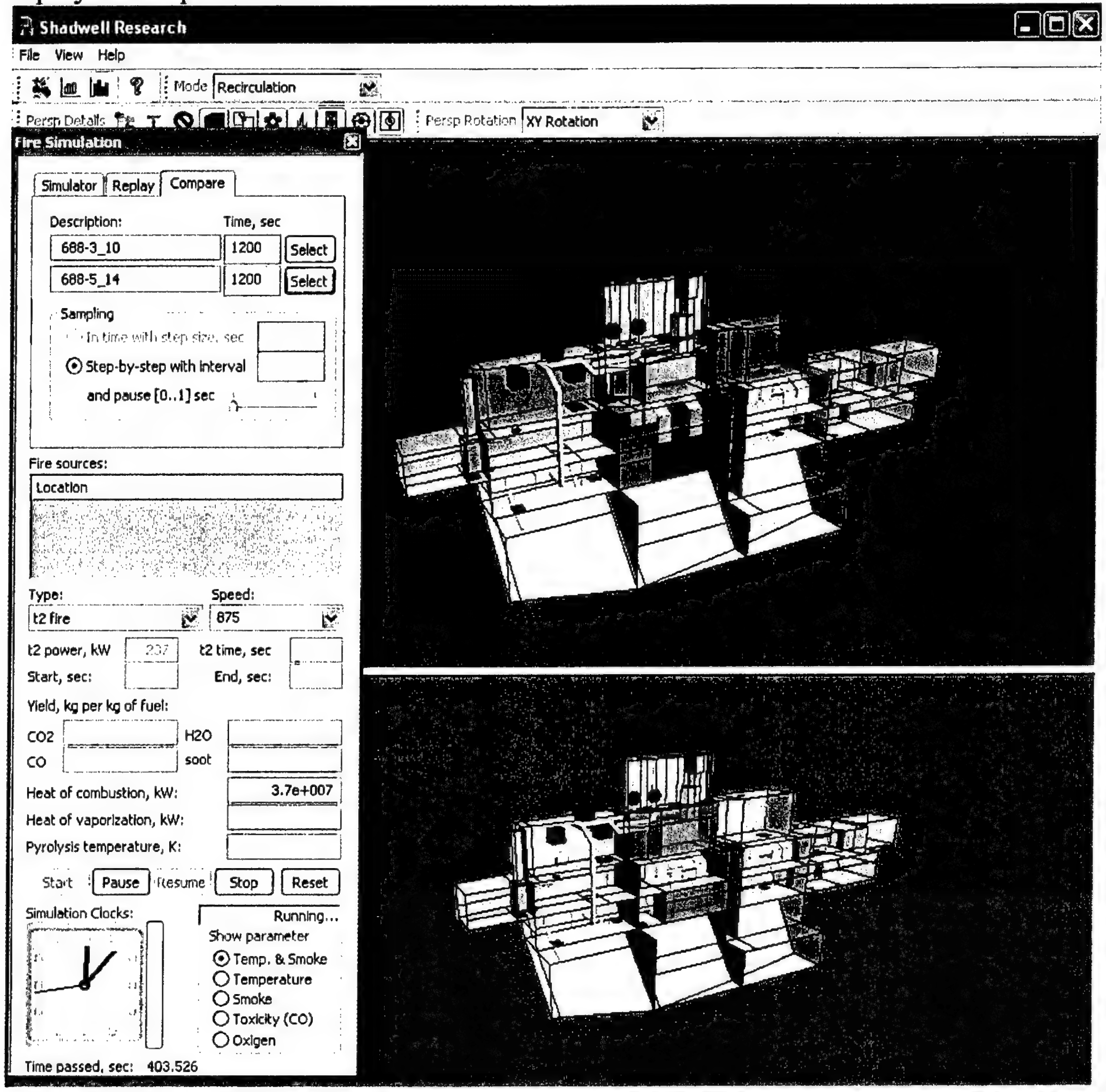
The replay dialog allows the user to control how the simulation is replayed. The replay (reading from a file) is much faster than real-time simulation. To slow down the replay the user may request a pause (a delay) between time steps. On the other hand, for very long simulations, the user may also request skipping time steps (e.g., displaying every second or every third time step) making the replay even faster.

The network model uses a variable time step during the simulation. While replaying, the user may select between constant rate of time steps (which corresponds to a variable temporal rate) and constant temporal rate (with variable amount of time steps skipped).

Finally, the user may skip the beginning of the simulation and jump directly to a specified time step or specified simulation time.

7.3.5 Comparison of Two Simulations

Back-to-back comparison of two simulations (Fig. 31) is an extension of the replay mode with two data sources and two ship models. Using the "Compare" tab of the Fire Simulation dialog window (Fig. 26) the user selects two previously run simulations and controls them the same way as it is in the case of a simple replay. The simulations are replayed in separate threads.



Mode: Recirculation
Persp Rotation: XY Rotation

Description:	Time, sec
688-3_10	1200
688-5_14	1200

Sampling
In time with step size, sec
[✓ Step-by-step with interval]
and pause [0..1] sec

Fire sources:
Location

Type: t2 fire
Speed: 875
t2 power, kW: 237
t2 time, sec:
Start, sec:
End, sec:
Yield, kg per kg of fuel:
CO2
H2O
CO
soot
Heat of combustion, kW: 3.7e+007
Heat of vaporization, kW:
Pyrolysis temperature, K:

Running...
Show parameter
[✓ Temp. & Smoke]
Temperature
Smoke
Toxicity (CO)
Oxigen

Time passed, sec: 403.526

Figure 31: Back-to-back comparison of two previously run simulations

8.0 VISUALIZATIONS OF THE SIMULATION RESULTS

The currently used Network model provides one value of each dependent variable (temperature, smoke, CO and oxygen concentration) per volume (i.e. ship compartment). This representation does not provide enough information for creating a quality value gradient in the scope of that volume. For example, if there were more than a single value for the smoke then it would be possible to visualize it as a non-homogeneous instance inside of compartments.

Nevertheless we can still get a picture that will decently reflect the processes taking place during the fire by using color maps, which is one of the best ways to represent physical values changing in time or space. Moreover, such data granularity is satisfying for real-time ship control and making appropriate decisions in case of emergency.

8.1 Color Mapping

The simulator produces a data block that contains several scalar output values – temperature, density of smoke (soot) or visibility, concentration of oxygen and, finally, concentration of toxic substances such as carbon monoxide.

A good representation of such data type is a color. According to studies in cognitive science, color saturation should be used to represent the magnitude of a scalar. Indeed, changing from light grey to dark grey indicates that a displayed parameter either increased or decreased in magnitude whereas changing from yellow to red indicates a qualitative transition. One exception is when one desires to show critical transitions, such as when a compartment becomes uninhabitable. For such cases, dramatic change in color hue vividly notifies the user about passing some important thresholds (Fig. 32).

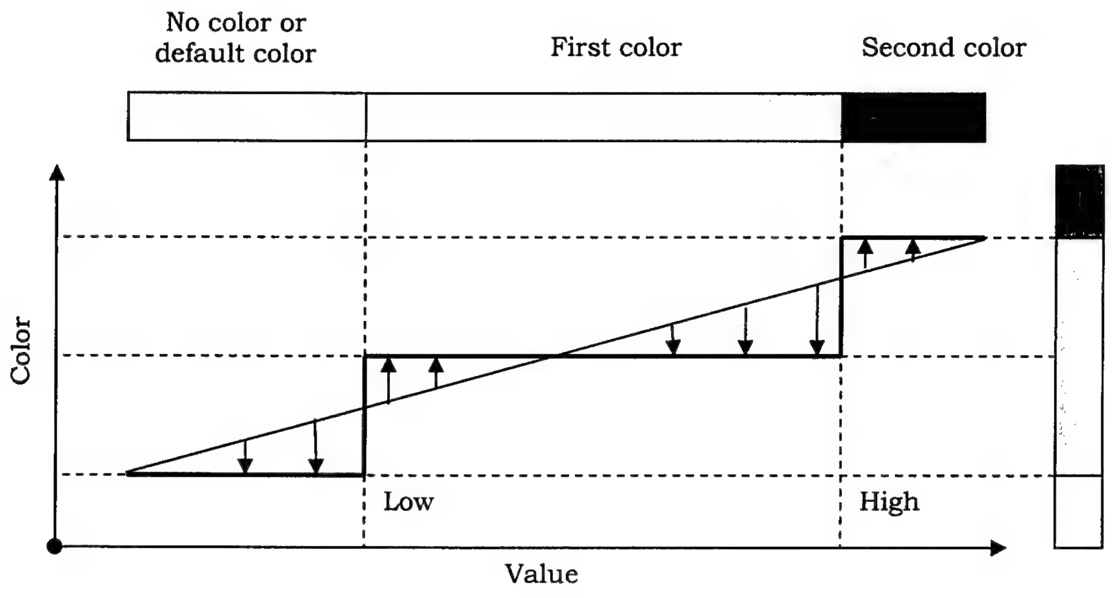


Figure 32: Color mapping with two critical levels and constant gradient.

This technique is not sufficient in all situations, in particular when this software is used as a tactical tool. A person who makes simulations or who controls a ship in real time also wants to know when a value is close to critical points in order to be prepared to take appropriate actions (for example, to give a command to put on protective suits). Therefore, the selection function should be modified so that in the end of each range (except for the last one) there will be a region showing a transition from a current value range to the next.

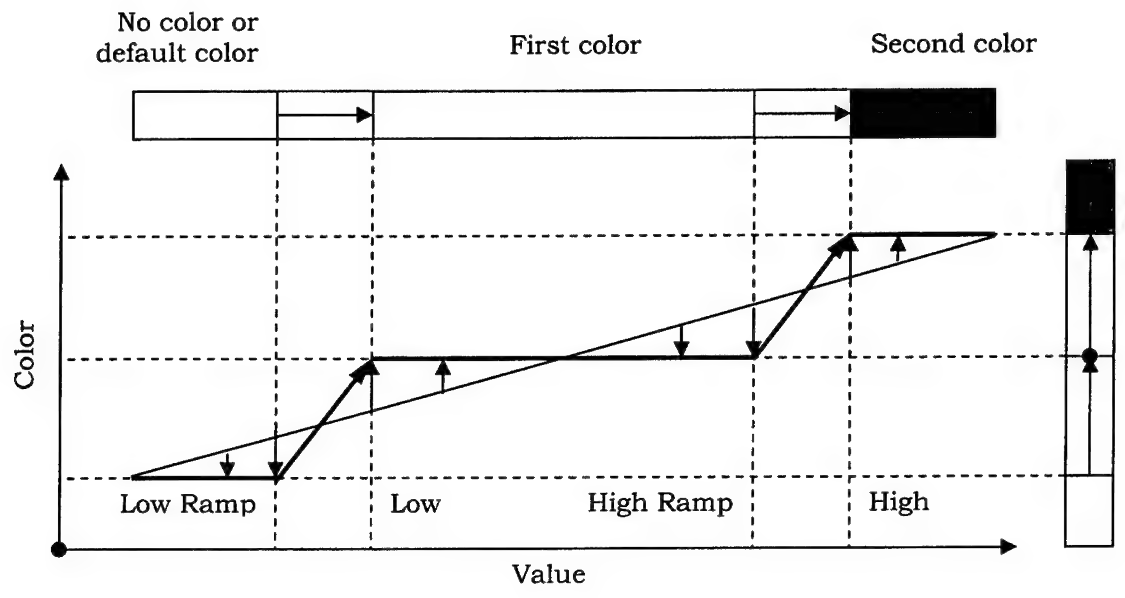


Figure 33: *Color mapping with two critical levels and two gradients at the end of each threshold.*

Thus, the color map is defined by four parameters (threshold values): low ramp, low, high ramp and high. The values below the low ramp threshold are ignored; that is, no visual effects are produced as the conditions do not impact activities of people. The values between the low ramp and the low thresholds correspond to the transition between a safe region and a region where persons need protective equipment. This is visualized by painting the compartment walls in yellow with intensity proportional to temperature or concentration of species. The region between the low and high ramp threshold is shown in saturated yellow (i.e., constant intensity). That is, in this region the color does not show any differences in temperatures or concentration of species as long as the values are in the range where persons with protective equipment are safe. The values above the high threshold are shown in saturated red and they correspond to inhabitable areas. Finally, the values between the high ramp and high threshold indicate the transition between the limited access and no access regions and are visualized by a change in hue from yellow to red, proportionally to the value.

The loss of visibility due to smoke is visualized by opacity rather than color (for details see below); nevertheless the same four threshold approach is taken: no visibility (< 2

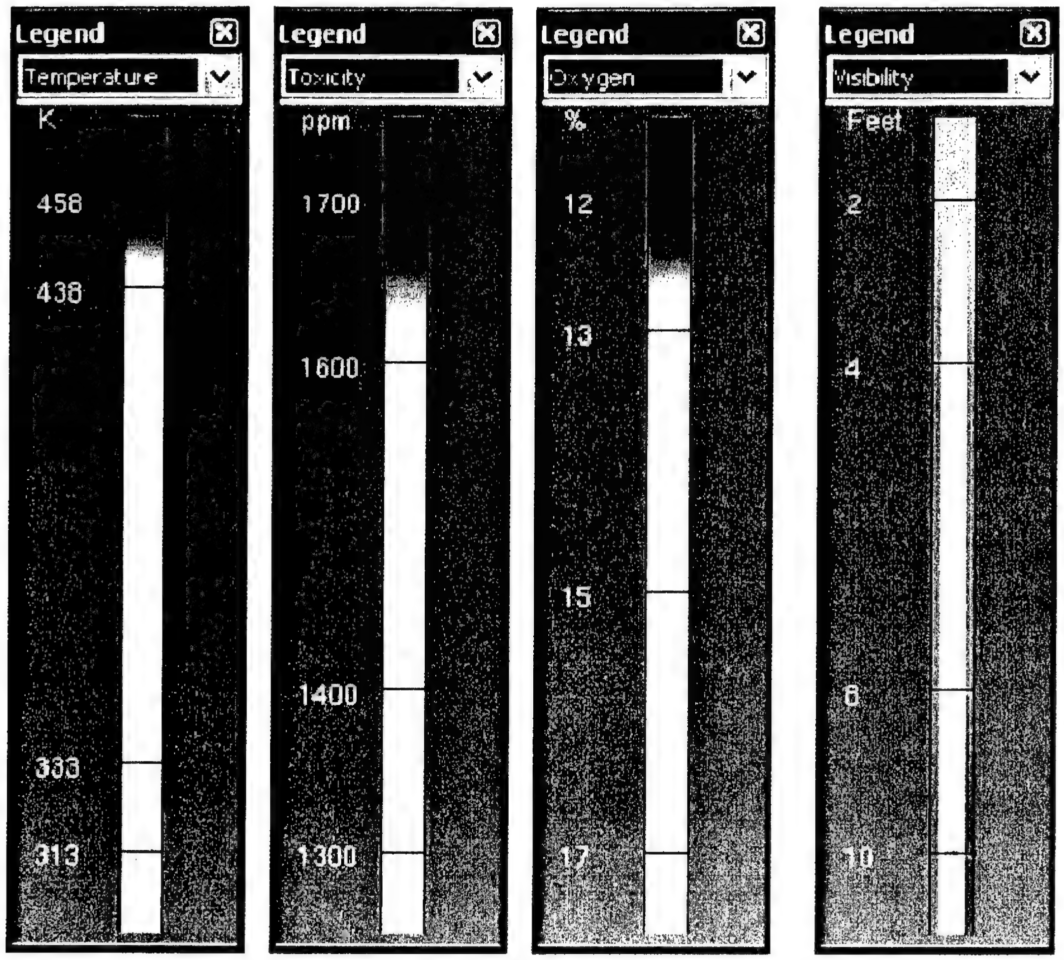


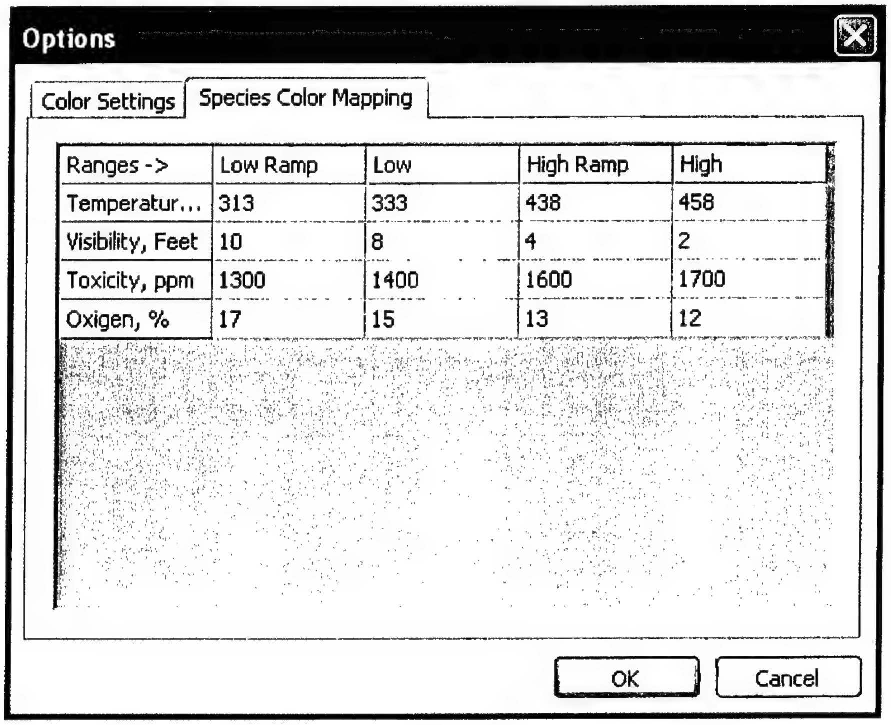
Figure 34: Color and opacity mapping for four variables: temperature, CO concentration, Oxygen level and visibility

1.1 Options Dialog Tab: Adjusting the Color Mapping

The advantage of the described method of color mapping is flexibility in adjusting of the threshold values. Indeed, the only thing to be done is to define the threshold values and provide a proper range identification mechanism to be able correctly normalize a parameter value in scope of that range.

There are three scalar value parameters simulated by the Network model – temperature, oxygen and toxicity (CO). Critical levels for each of them were recommended by Hughes Associates, Inc., but the user also has ability to change them through the *Species Color Mapping* tab in the *Options* dialog window (Fig. 35).

There are three scalar value parameters simulated by the Network model – temperature, oxygen and toxicity (CO). Critical levels for each of them were recommended by Hughes Associates, Inc., but the user also has ability to change them through the *Species Color Mapping* tab in the *Options* dialog window (Fig. 35).



Ranges ->	Low Ramp	Low	High Ramp	High
Temperatur...	313	333	438	458
Visibility, Feet	10	8	4	2
Toxicity, ppm	1300	1400	1600	1700
Oxigen, %	17	15	13	12

Figure 35: *The Option dialog window for adjusting threshold values for temperature, visibility, CO concentration and oxygen level*

8.3 Smoke

The smoke is perceived as a loss of clearness of details of objects. There can be different ways of achieving this effect.

First, using particle systems or volumetric smoke can produce the most realistic smoke. Although results are very persuasive, the degree of rendering complexity is very high. Each particle is represented as an individual object, so for very dense smoke the number of particles must be rather high.

The second approach that can be used is imposing another semi-transparent object in front of the details to be obscured. In this case, controlling the amount of smoke reduces to manipulating the color's alpha channel. There is less smoke when an alpha value is lower and vice versa. There is a good reason for using this method – the data block produced by the simulator is very sparse; that is, it has just a single data value for each compartment. Such conditions prevent a quality smoke analysis inside each compartment, so making complex smoke representations with particle system is hardly possible and even redundant.

The compartment is represented as a set of sides or quadrangles. Thus, the effect of a smoky room is achieved by drawing the same compartment over again with a side color different from the original in its alpha channel value. An alpha blending produces desired results as shown in Fig. 36.

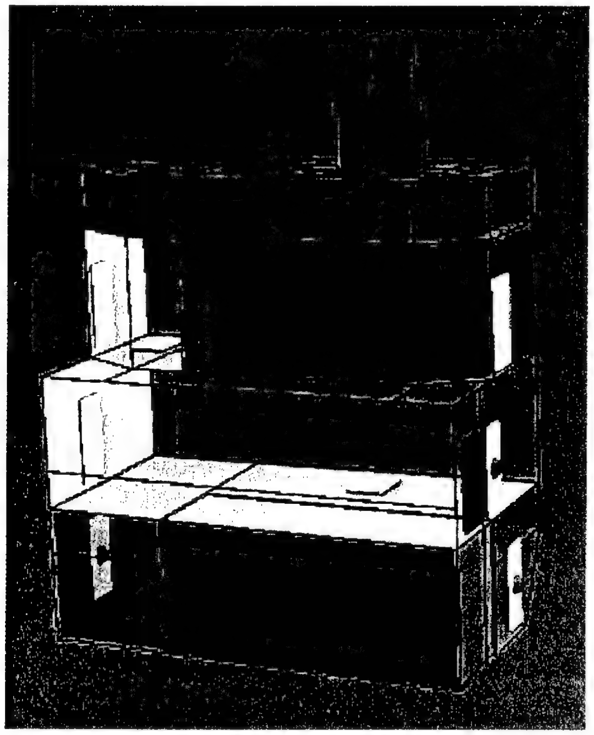


Figure 36: *Smoke visualization: the top compartments and the bottom left are partially smoked; the compartment in the middle is free of smoke; a bottom front compartment has a higher concentration of smoke than any.*

9.0 SUMMARY

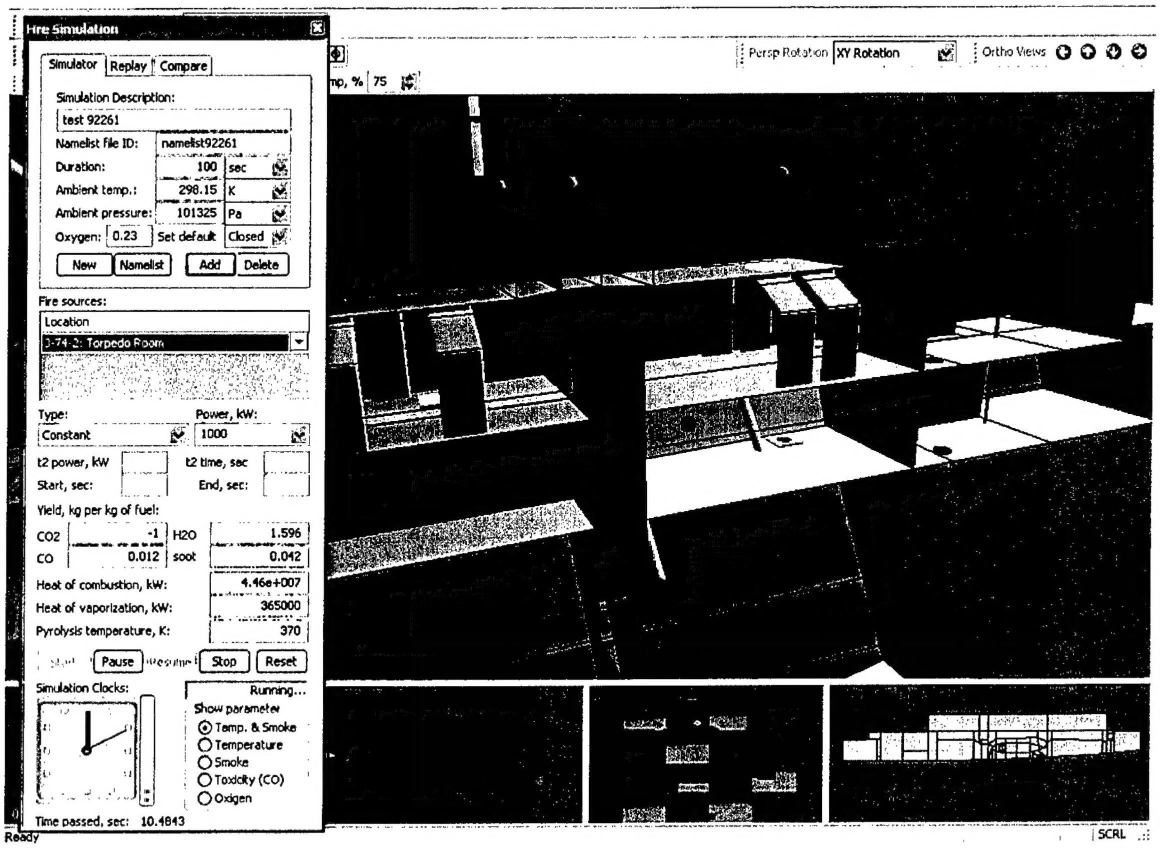
In collaboration with NRL, Hughes Associates ,Inc. and Havlovick Engineering Services, we have developed a simulation environment for an onboard fire network model. The complete system (3D model, input GUI, the network model developed by HAI, database, the runtime environment and visualizations) allow the user to

- specify the state of the ship (in particular, the state of the active elements of the ventilation system)
- specify the location and parameters of a fire
- run the simulation of fire and smoke spread

- display the results in a graphical form in real time (Fig 37)
- replay and compare previously run simulations

Currently the system simulates fires on board the ex-USS Shadwell/test area 688 emulating a submarine. The results have been validated by Hughes Associates, Inc. against data coming from the actual fire tests performed on board the ex-USS *Shadwell.*

The system is a proof of concept prototype and the future work will transform it as an onboard tactical tool, as well as tool for design support and training.



Simulation Description: test 92261
Namelist file ID: namelist92261
Duration: 100 sec
Ambient temp.: 298.15 K
Ambient pressure: 101325 Pa
Oxygen: 0.23 Set default Closed

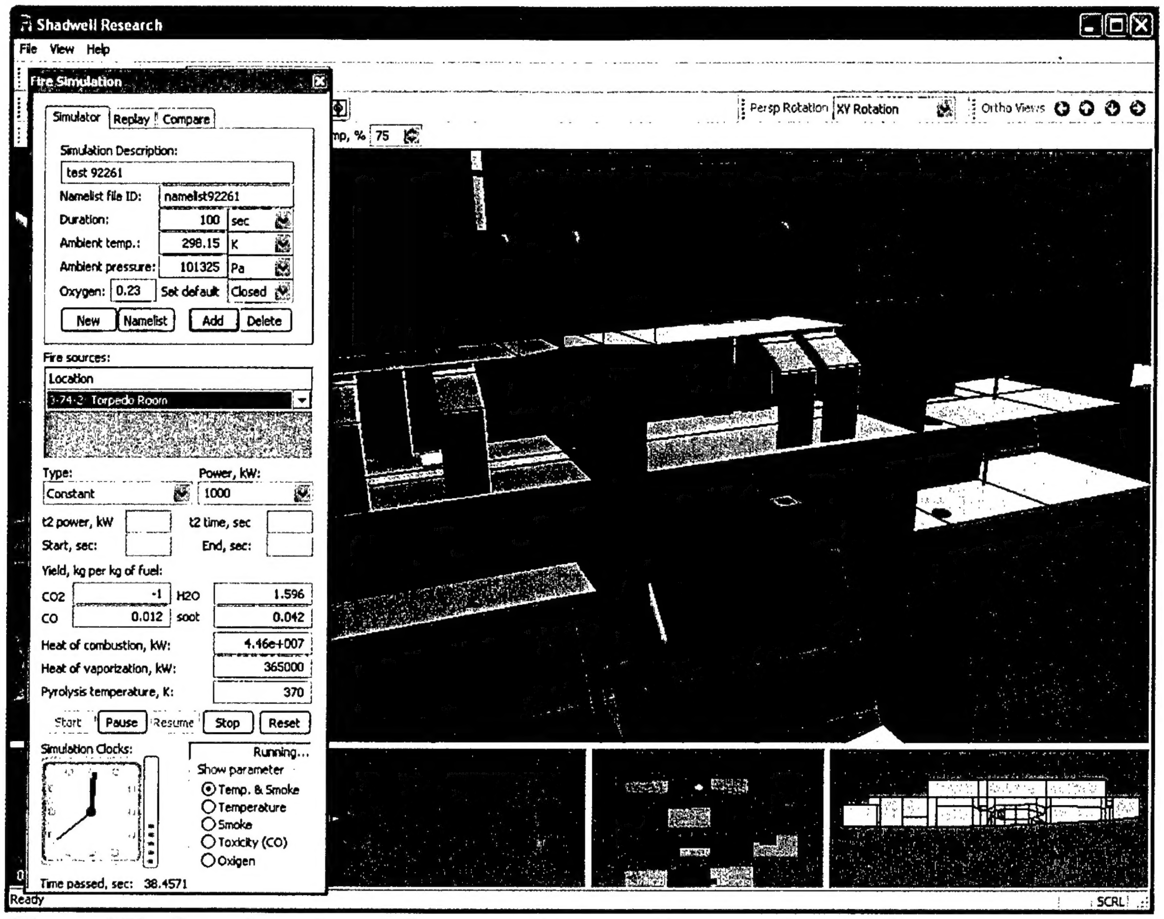
Fire sources:
Location
3-74-2: Torpedo Room

Type: Constant
Power, kW: 1000
t2 power, kW
t2 time, sec
Start, sec:
End, sec:

Yield, kg per kg of fuel:
CO2 -1 H2O 1.596
CO 0.012 soot 0.042
Heat of combustion, kW: 4.46e+007
Heat of vaporization, kW: 365000
Pyrolysis temperature, K: 370

Simulation Clocks:
Running...
Show parameter
[✓ Temp. & Smoke]
Temperature
Smoke
Toxicity (CO)
Oxigen
Time passed, sec: 10.4843

Figure 37a: Simulation of a fire of a constant power of 1000 kW set in the torpedo room (symbolically marked with a red ball): a snapshot taken at 10.4843 seconds after the fire started. The fire has no effect on any compartments except for the torpedo room, where the temperature requires wearing protective gear. No substantial smoke at this time.



Simulation Description: test 92261
Namelist file ID: namelist92261
Duration: 100 sec
Ambient temp.: 298.15 K
Ambient pressure: 101325 Pa
Oxygen: 0.23 Set default Closed

Fire sources:
Location
3-74-2 Torpedo Room

Type: Constant
Power, kW: 1000
t2 power, kW
t2 time, sec
Start, sec:
End, sec:

Yield, kg per kg of fuel:

CO2	-1	H2O	1.596
CO	0.012	soot	0.042

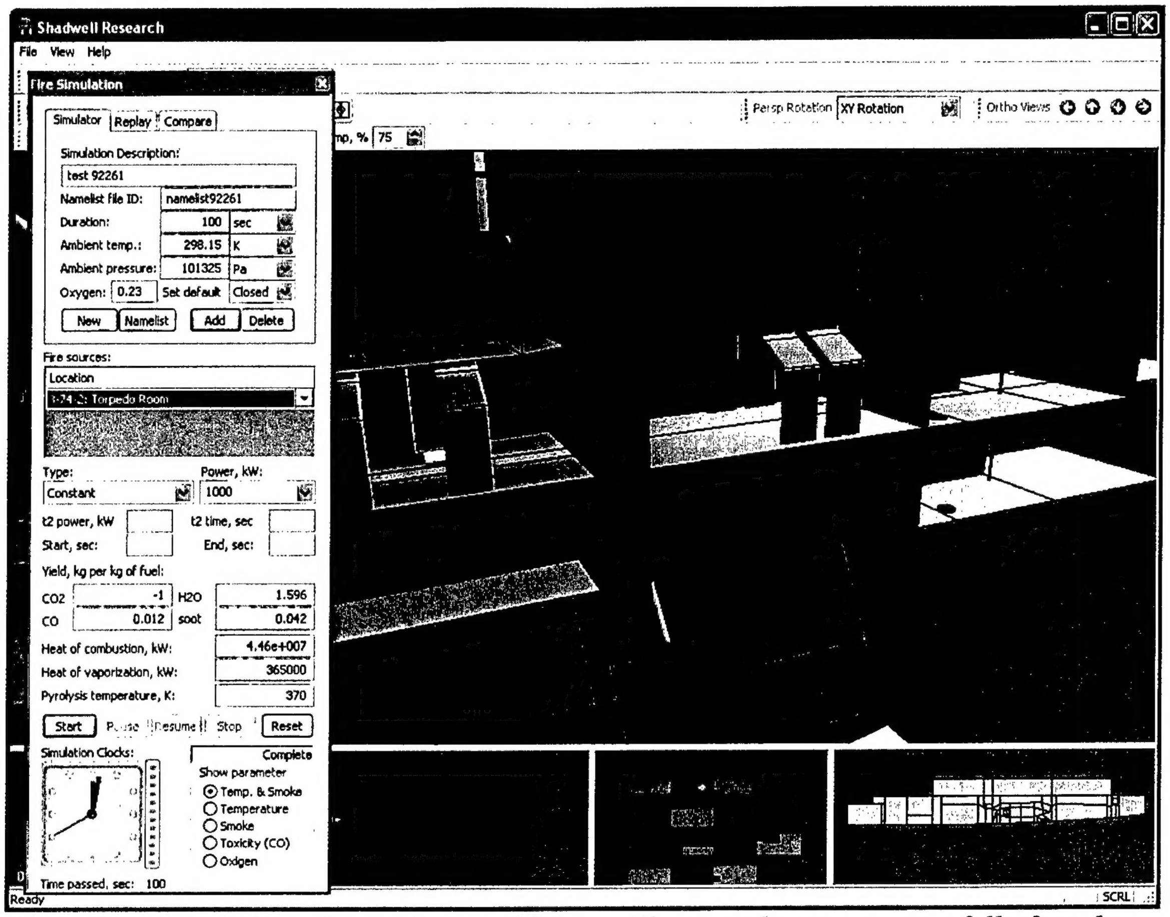
Heat of combustion, kW: 4.46e+007
Heat of vaporization, kW: 365000
Pyrolysis temperature, K: 370

Simulation Clocks: Running...
Show parameter
[✓ Temp. & Smoke] Temperature Smoke Toxicity (CO) Oxigen

Time passed, sec: 38.4571

mp, % 75

Figure 37b: *Snapshot of same fire at 38.4571 sec. The torpedo room becomes uninhabitable. The heat propagates through the frame bays to the combat systems room, which now requires protective gear.*



Simulation Description: test 92261
Namelist file ID: namelist92261
Duration: 100 sec
Ambient temp.: 298.15 K
Ambient pressure: 101325 Pa
Oxygen: 0.23 Set default Closed

Fire sources:
Location
1-74-2: Torpedo Room

Type: Constant
Power, kW: 1000
t2 power, kW
t2 time, sec
Start, sec:
End, sec:

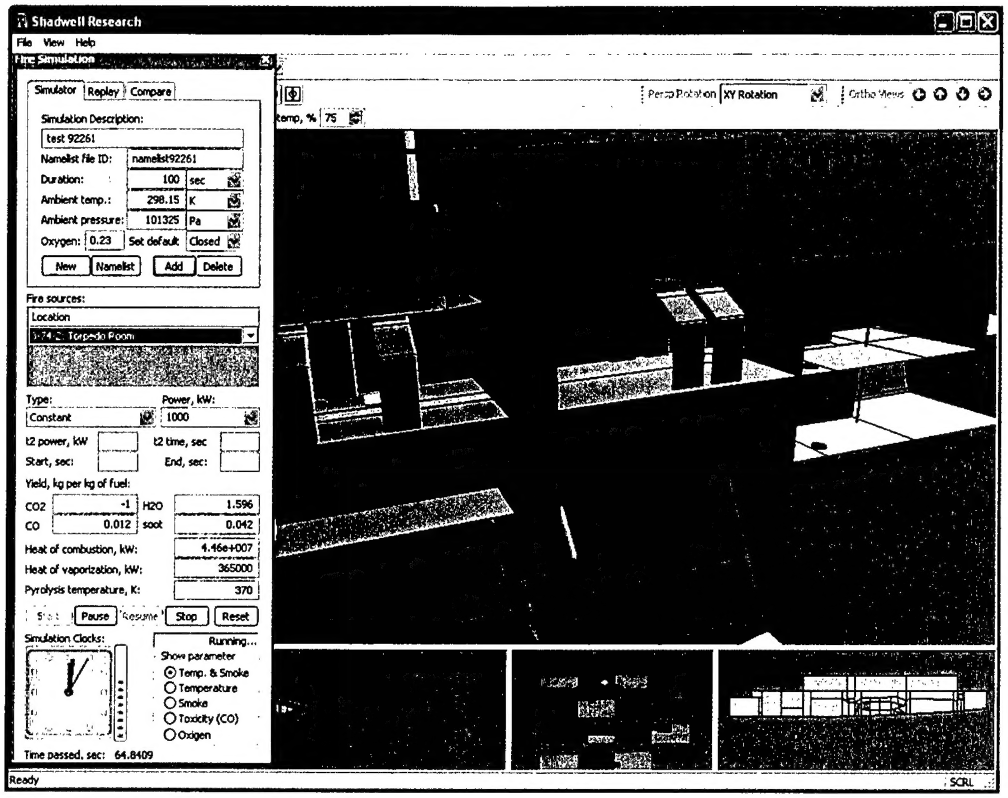
Yield, kg per kg of fuel:
CO2 -1 H2O 1.596
CO 0.012 soot 0.042

Heat of combustion, kW: 4.46e+007
Heat of vaporization, kW: 365000
Pyrolysis temperature, K: 370

Simulation Clocks:
Complete
Show parameter
[✓ Temp. & Smoke]
Temperature
Smoke
Toxicity (CO)
Oxygen
Time passed, sec: 100

XY Rotation
Temp, % 75

Figure 37c: *Snapshot of same fire at 100.0 sec. The torpedo room is now full of smoke. The heat and smoke propagates to the combat system room, resulting in limited visibility there. The heat propagates further to the control room (through the joiner doors) and the crew living room (through the hatch).*



Simulation Description: test 92261
Namelist file ID: namelist92261
Duration: 100 sec
Ambient temp.: 298.15 K
Ambient pressure: 101325 Pa
Oxygen: 0.23 Set default Closed

Fire sources:
Location
3-74-2: Torpedo Room

Type: Constant
Power, kW: 1000
t2 power, kW
t2 time, sec
Start, sec:
End, sec:

Yield, kg per kg of fuel:
CO2 -1 H2O 1.596
CO 0.012 soot 0.042
Heat of combustion, kW: 4.46e+007
Heat of vaporization, kW: 365000
Pyrolysis temperature, K: 370

Simulation Clocks:
Running...
Show parameter
[✓ Temp. & Smoke]
Temperature
Smoke
Toxicity (CO)
Oxygen
Time passed, sec: 64.8409

Figure 37d: *Another simulation with identical fire setting as in Figure 37a-c. The only difference is that the joiner door between the torpedo room and the store room is now open and the hatch in the torpedo room is closed. The propagation of the heat from the torpedo to the store room is clearly seen. There is no visible effect of the status of the hatch. The snapshot is taken at 64.8409 seconds after the fire started.*

10.0 ACKNOWLEDGEMENTS

This material is based upon work supported by GEO-CENTERS, INC. Subcontract 40939MK-GC-3448 under NRL Contract N00173-00-C-2096.

11.0 REFERENCES

[1] Dmitry Shulga, "The Simulation System for Propagation of Fire and Smoke", Master Thesis, Mississippi State University, 2003. Also available from http://www.erc.msstate.edu/~haupt/FireSmoke/dmitry.pdf

[2] Floyd, J., Hunt, S., Williams, F., and Tatem, P., "Fire and Smoke Simulator:(FSSIM) Version 1 - Theory Manual", NRL/MR/6180—04--, in publication

[3] Floyd, J., Hunt, S., Williams, F., and Tatem, P., "Fire and Smoke Simulator(FSSIM) Version 1. - User's Guide", NRL/MR/6180—04--, in publication